



Sistemi per Basi di Dati

Franco Scarselli



Argomenti trattati durante il corso

- Tecnologia delle basi di dati
 - Organizzazione fisica dei DBMS
 - Gestione delle Interrogazioni
 - Gestione delle transazioni
 - Database distribuiti



Argomenti trattati durante il corso II

- Evoluzione delle basi di dati
 - Basi di dati e Web
 - Basi di dati per il supporto delle decisioni
 - Data Mining



Regole

- Il corso prevede
 - una prove finale (scritta?)
 - Un piccolo progetto finale
- Le prove finale può essere recuperata in qualsiasi appello che verrà tenuto in forma orale
- Per gli appelli successivi al primo si richiede di contattare preliminarmente il docente



Libri di testo

Il materiale da studiare per il corso è contenuto nelle slide fornite dal docente. I seguenti libri possono essere utili per eventuali approfondimenti, ma non sono indispensabili.

1. **Basi di Dati: Architetture e linee di evoluzione**
P. Atzeni, S. Ceri, P. Fraternali, S. Paraboschi, R. Torlone
McGraw-Hill, 2003
2. **Database Systems: The Complete Book**
H. Garcia Molina, J. D. Ullman, J. Widom
Prentice Hall, 2001
3. **Database System Implementation**
H. Garcia Molina, J. D. Ullman, J. Widom
Prentice Hall, 2000 (2 contiene una versione aggiornata di 3)



Libri di testo II

4. **Database Systems: a Practical Approach to Design, Implementation, and Management (3th Edition)**
T. Connolly, C. Begg
Addison Wesley, 2002
5. **Database Management Systems (2nd Edition)**
R. Ramakrishnan, J. Gehrke
McGraw-Hill, 2000

La struttura interna dei DBMS



Caratteristiche dei DBMS

Devono garantire

- **Affidabilità**

(Non si perdono informazioni in caso di malfunzionamento)

- **Privatizza**

(Restrizione degli accessi)

- **Efficienza**

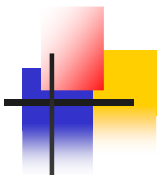
(Il sistema svolge inserimenti, modifiche, ricerche in un tempo ragionevole)

Gestiscono **collezioni di dati**:

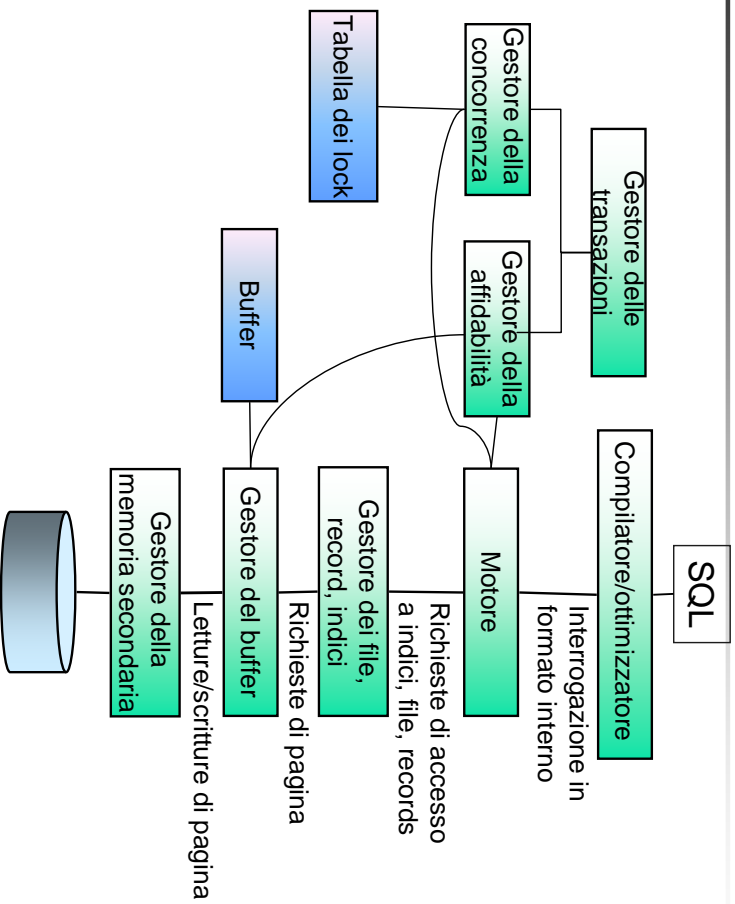
- **Grandi**

- **Persistenti**

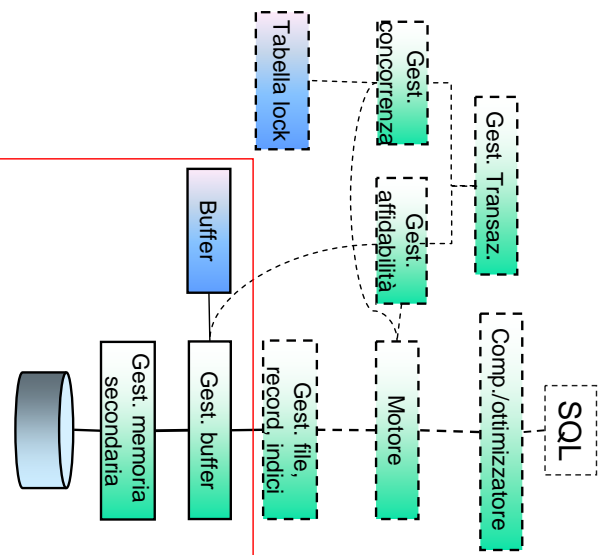
- **Condivise**



Le componenti di un DBMS



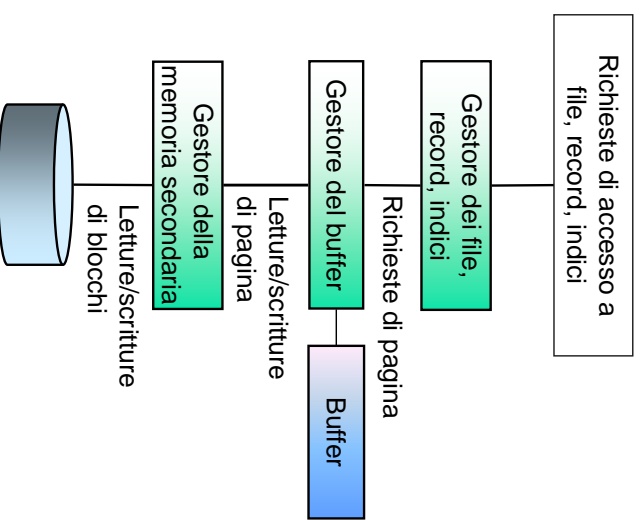
Gestione della memoria secondaria



Gestione della memoria secondaria

Le componenti coinvolte

- **Gestore dei file, record, indici**
Analizza le richieste di accesso a record, file e indici e individua le pagine da caricare
- **Gestore dei buffer**
Gestisce una cache dei record contenuti nella memoria secondaria
- **Gestore della memoria secondaria**
Scrive e legge effettivamente la memoria secondaria, individuandone la collocazione sui dischi



Memoria secondaria (dischi)

Alcuni fatti noti

- La memoria secondaria è organizzata in blocchi (pagine)
 - Un blocco è l'unità minima trasferibile
 - Un blocco va da pochi KByte a alcune decine di KByte
- L'accesso alla memoria secondaria
 - tempo di **posizionamento della testina** (5-50ms)
 - tempo di **latenza** (5-10ms)
 - tempo di **trasferimento** (0.5-2ms)
 - in media circa 10 ms



Memoria secondaria (dischi) II

L'accesso alla memoria secondaria

- È estremamente più costoso dell'accesso alla memoria primaria
- Il tempo di accesso alla memoria secondaria dipende dall'ordine in cui i blocchi vengono letti/scritti
 - La lettura di blocchi contigui può costare **10/100 volte meno**
- Con i DBMS, spesso il tempo speso in accesso alla memoria primaria è' ininfluente:
 - la complessità delle operazioni si può misurare in termini di **numero di accessi alla memoria secondaria**

Per questo motivo

- **Gli algoritmi e le strutture date usati nei DBMS possono essere molto diverse da quelle tradizionali!!**



Un esempio: l'ordinamento

Cosa succede quando la memoria primaria non contiene i dati?

Per l'ordinamento si utilizza un algoritmo divide et impera

- Si suddividono i dati in sottoparti, in modo che ogni parte sia contenuta in memoria primaria
- si cerca di minimizzare gli accessi alla RAM

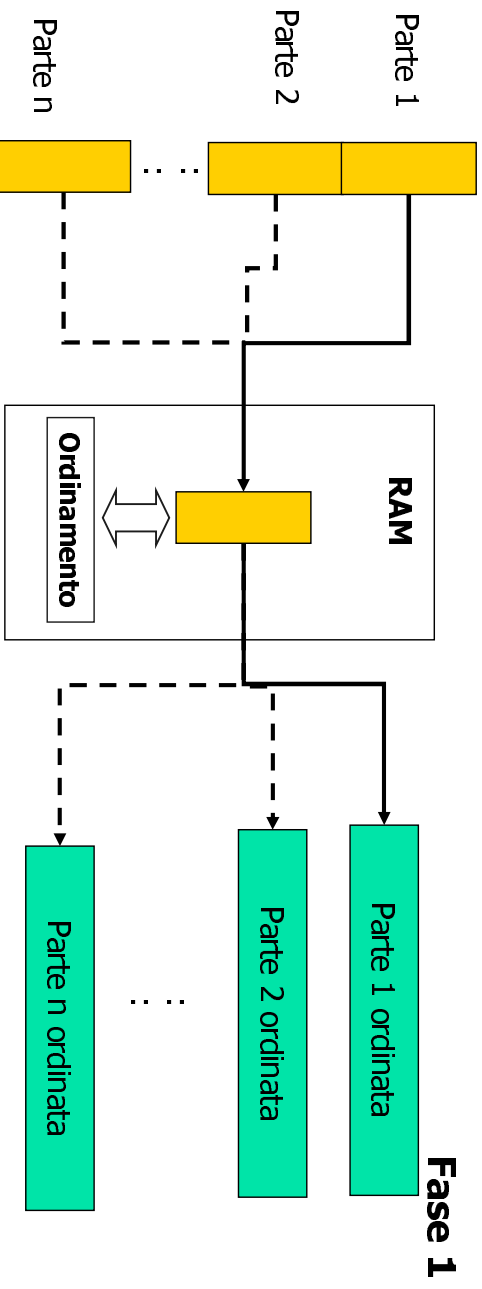
Two phase **multiway merge-sort**

- I dati sono divisi in parti della dimensione della RAM disponibile
- nella prima fase ogni parte è caricato in RAM, ordinata (ad. es. con quicksort) e riscritta in memoria secondaria
- nella seconda fase le parti sono unite insieme (merge) caricando i dati in RAM blocco per blocco

Un esempio: l'ordinamento II

Osservazioni

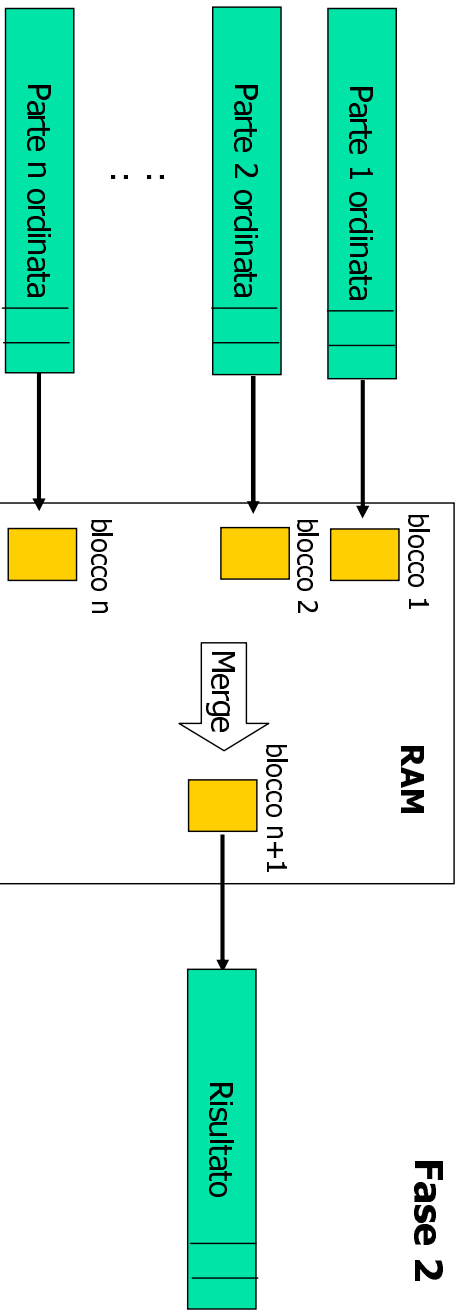
- Il file da ordinare si suddivide in parti della dimensione della RAM
- Si ordinano le parti una alla volta



Un esempio: l'ordinamento III

Osservazioni

- Si carica un blocco per ogni parte e si selezionano i record maggiori
- Ogni blocco viene caricato in memoria solo due volte



Un esempio: l'ordinamento IV

La complessità del two phase multiway merge-sort

- Richiede $O(4 B(F))$ accessi alla memoria secondaria
 - $B(F)=n$ indica il numero dei blocchi del file
- Serve se $B(F) > M$
 - M indica i blocchi disponibili in memoria principale
- Funziona se $B(F) < M^2$
- Si ricorre ad algoritmi a più fasi se $B(F) > M^2$

Esempio

- Blocchi 64KB, RAM 1G
- Si può calcolare il numero dei blocchi della Memoria $M=2^{14}=16384$
- Se un file è più piccolo di 1G si applica algoritmi ad un passo
- Se un file è più grande di $M^2=2^{28}=268.435.456$ blocchi si usano algoritmi a 3 passi
($M^2 * 64K = 2^{44}=16Tera$)

Migliorare l'accesso alla memoria secondaria

Soluzioni per migliorare l'accesso alla memoria secondaria

- Organizzare i dati sui dischi
 - dati correlati sono messi sullo stesso cilindro ed, eventualmente in blocchi contigui
 - vantaggioso se il modo in cui si accede ai dischi è prevedibile (es. fase 1 dell'ordinamento)
 - non utile in caso in cui il modo di accedere ai dischi sia imprevedibile (es. fase 2 dell'ordinamento o molti processi contemporanei)
- Riordinare le richieste di lettura/scrittura
 - un algoritmo implementato dal controller o dal sistema operativo ordina le richieste di lettura/ scrittura (ad. es. l'algoritmo dell'ascensore)
 - riduce i costi di accesso per qualsiasi applicazione
 - valido se ci sono molte richieste di lettura scrittura per le quali i tempi di attesa sono lunghi

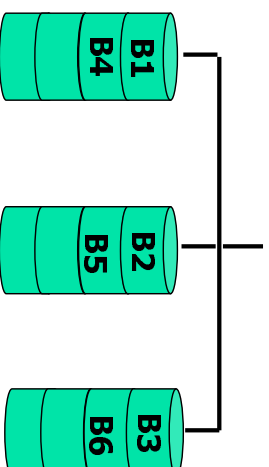
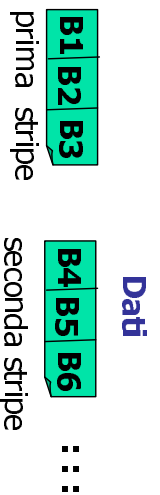
Migliorare l'accesso alla memoria secondaria: dischi multipli

■ Dischi RAID

- i dati sono partizionati in unità di uguale lunghezza chiamate **stripe** e distribuiti su n dischi (ad esempio con una strategia round robin): **le stripe** possono essere scritte e lette in parallelo

■ RAID 0

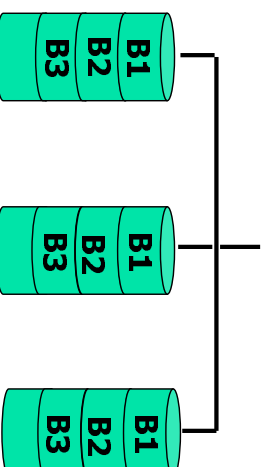
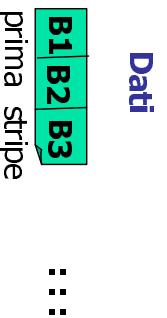
- i dati sono distribuiti su entrambi i dischi
- aumento di velocità di lettura e scrittura (per un fattore n), anche nel caso in cui i blocchi siano acceduti con un ordine imprevedibile
- diminuzione dell'affidabilità (MTBF) di un fattore n



Migliorare l'accesso alla memoria secondaria: dischi multipli II

■ RAID 1 (mirroring)

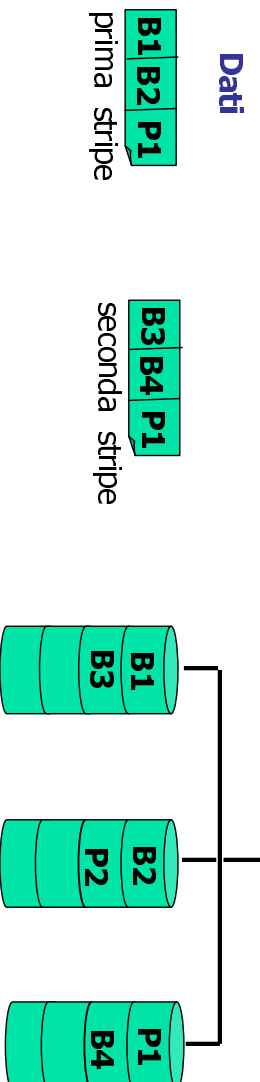
- i dati sono duplicati e una copia viene memorizzata su ogni disco
- aumento di velocità di lettura, ma non di scrittura
- aumento dell'affidabilità di un fattore n



Migliorare l'accesso alla memoria secondaria: dischi multipli III

■ RAID 5

- ogni stripe è costituita da $n-1$ blocchi più un blocco di parità:
 - i blocchi vengono distribuiti su dischi diversi
 - i blocchi di parità permettono di ricostruire le stripe in caso di malfunzionamento di uno dei fischi
- aumento di velocità di lettura e scrittura all'incirca di un fattore $n-1$ (ad eccezione di scritture o letture di blocchi sullo stesso disco)
- aumento dell'affidabilità all'incirca di un fattore $n-1$ (anche se più dischi insieme tendono a riscaldarsi di più, a rompersi insieme,)



Migliorare l'accesso alla memoria secondaria: dischi multipli IV

■ Dischi diversi per strutture dati diverse

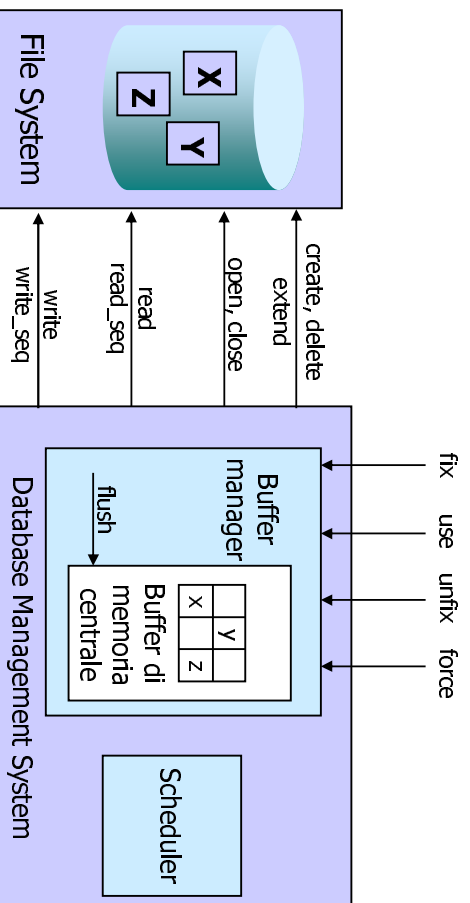
- Il sistemista decide come distribuire le tabelle, gli indici e i log
- L'aumento delle prestazioni e dell'affidabilità dipende dal particolare database

Migliorare l'accesso alla memoria secondaria: buffer e prefetch

- **Buffer e Prefetch**
 - si cerca di prevedere i blocchi da caricare/scrivere e si mettono nel buffer
 - vantaggioso se sono prevedibili i blocchi richiesti ma non il momento in cui verranno richiesti (es. fase 2 dell'algoritmo di ordinamento)
 - richiede maggiore spazio per il buffer

Gestione dei buffer

- **Buffer:** zona di memoria centrale preallocata e condivisa fra le transazioni





Organizzazione del buffer

- Il buffer è organizzato in pagine

- La dimensione di una pagina è un multiplo della dimensione dei blocchi di ingresso-uscita utilizzati nelle letture/scritture sui dispositivi di memoria di massa
- Dimensioni tipiche delle pagine variano fra 2Kb e 64Kb
- La velocità di accesso ai record di pagine nel buffer può essere 10^6 volte maggiore rispetto alla memoria secondaria
- Le politiche di gestione si basano sul principio di **località dei dati**
 - i dati referenziati di recente hanno maggiore probabilità di essere referenziati nel futuro
 - sono simili a quelle usate dai sistemi operativi per gestire la memoria secondaria



Organizzazione del buffer II

- Il **gestore del buffer** gestisce il trasferimento delle pagine fra la memoria principale e la memoria di massa
 - riceve richieste di lettura e scrittura che esegue accedendo alla memoria secondaria solo quando necessario
 - mette a disposizione alcune primitive: **fix**, **unfix**, **use**, **force**, **flush**,...
- Le strutture dati del gestore del buffer includono una tabella che per ogni pagina indica
 - il corrispondente blocco fisico
 - se la pagina e' in uso o meno
 - se la pagina e' stata modificata



Primitive

- **fix**

- si richiede l'accesso ad una pagina che viene caricata nel buffer
- restituisce il riferimento alla pagina nel buffer
- la pagina risulta **valida** e allocata alla transazione
- la lettura da disco è richiesta solo se la pagina non era già presente nel buffer

- **use**

- viene usata dalla transazione per accedere alla pagina caricata
- conferma l'allocazione della pagina nel buffer e lo stato di **valida**



Primitive

- **unfix**

- indica che la transazione ha terminato di usare la pagina
- la pagina passa nello stato di **non valida**

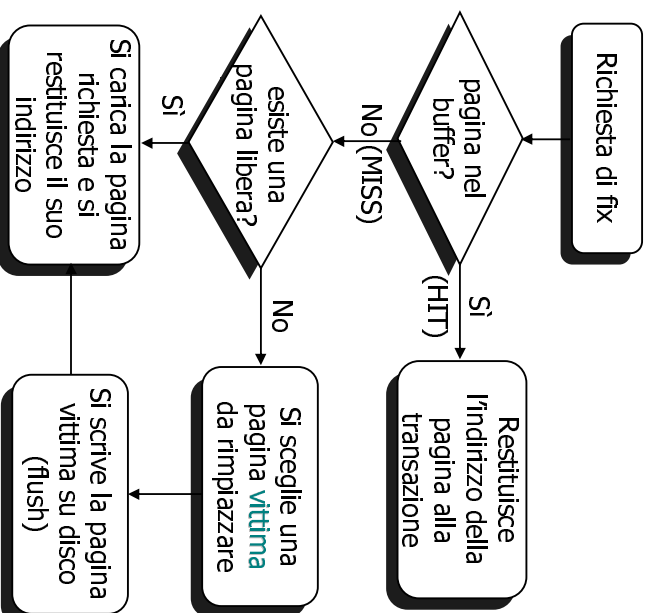
- **force**

- scrive una pagina in memoria di massa
- la scrittura è **sincrona** con la richiesta e le transazione si sospende fino a che non è terminata l'esecuzione della primitiva

- **flush**

- scrive su disco le pagine non più valide e inattive da più tempo
- la scrittura è **asincrona** e indipendente dalle transazioni
- rende **libere** le pagine del buffer salvate su disco
- può essere decisa dal gestore del buffer per **migliorare l'efficienza**

Esecuzione di fix



Politiche di gestione

■ Scelta della pagina vittima

- **steal**
 - Si possono selezionare anche le pagine attive di un'altra transazione
 - le pagine possono essere scritte prima del termine della transazione
 - può essere necessario recuperare il valore iniziale nel caso di un abort

- **no-steal (*)**

■ Scrittura delle pagine

- **force**

Le pagine attive di una transazione sono scritte in sincrono col commit
- **no-force (*)**

La scrittura dipende dalle decisioni del buffer manager (asincrona)

■ Pre-fetching/pre-flushing



DBMS e file system

Il DBMS gestisce la memoria secondaria, ma il sistema operativo gestisce file system: come interagiscono ??

- **Soluzione 1**

Il DBMS usa un file per ogni tabella e ogni indice

- Eventualmente, anche la gestione del buffer può essere lasciata al sistema operativo

- **Soluzione 2**

Il DBMS usa alcuni file

- L'allocazione dei dati all'interno dei file è gestita dal DBMS
 - il DBMS decide come allocare i blocchi dei file e come allocare i record all'interno dei file
 - spesso i DBMS usano un solo file
- La creazione e l'allocazione dei file sono gestite dal sistema operativo



DBMS e file system

- **Soluzione 3**

Una zona del disco viene allocata al DBMS

- Il DBMS gestisce autonomamente la zona allocata
- il DBMS può allocare in modo contiguo dati che accede in maniera sequenziale,

Vantaggi e svantaggi

- soluzione 3 implica **massima efficienza**, soluzione 1 minima efficienza
- soluzione 3 implica **massima affidabilità**, soluzione 1 minima affidabilità
- soluzione 1 implica **minimo costo di sviluppo** del DBMS
- la maggior parte dei DBMS usano soluzione 2, ma permettono anche soluzione 3
- man mano che i sistemi operativi diventano più efficienti e affidabili si va verso soluzione 1

Organizzazione dei dati

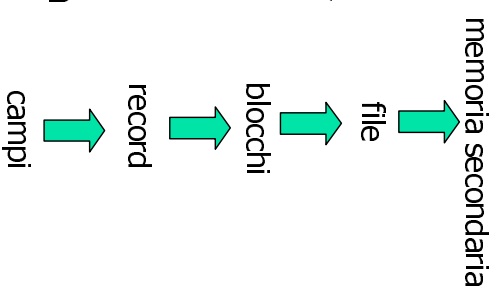
I dati sono organizzati in una gerarchia

- **campi**: contengono dati elementari (char, integer,...) rappresentati come noto
- **record**: una riga di una tupla
- **blocchi**: unità minima leggibile dalla memoria secondaria, assumeremo blocco=pagina

- **file**: **non corrisponde** necessariamente al file del file system. È un insieme di dati correlati

Per ogni elemento della gerarchia

- occorre scegliere la struttura dati da usare per allocarlo in memoria secondaria (all'interno dell'elemento figlio)
- ogni DBMS ha le proprie soluzioni: di seguito vediamo alcuni esempi



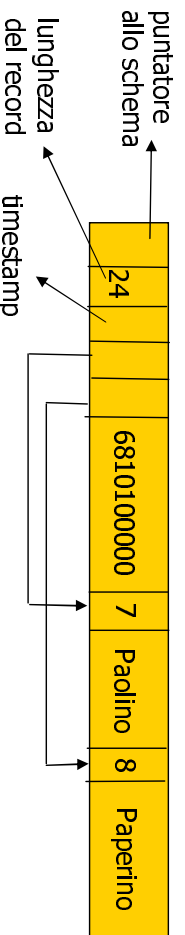
I campi nei record

Insieme ai dati del record si può memorizzare un **header**

- puntatore allo schema
- lunghezza del record
- timestamp
- altre informazioni

Osservazioni

- l'header **può essere omesso** se tutti i record di uno schema appartengono alla stessa tabella
- l'header contiene **puntatori all'inizio dei campi** a dimensione **variabile**



```
CREATE TABLE studenti(  
  matricola CHAR(9),  
  nome VARCHAR(15),  
  cognome VARCHAR(15)  
)
```

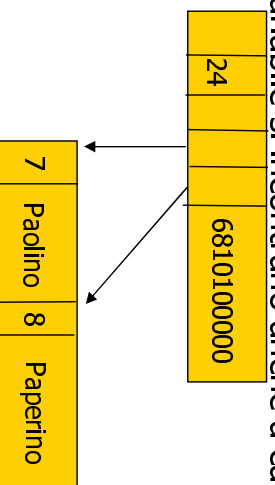
I record a dimensione variabile

I campi a dimensione variabile possono essere **memorizzati anche in un altro blocco**

- il record è piu' piccolo: ricerche piu' veloci
- accesso al campo variabile piu' lento
- compromesso: campi a dimensione variabile memorizzati in un altro blocco, ma sullo stesso cilindro
- adatta per campi grandi

In SQL 3, record a dimensione variabile si incontrano anche a causa di

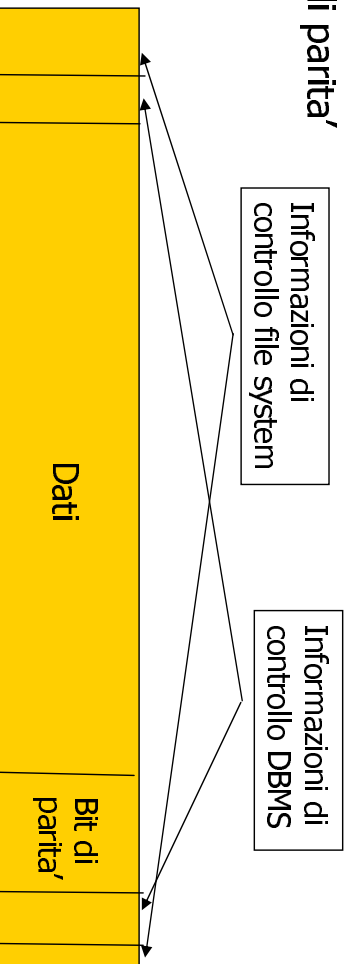
- campi multipli
- record a formato variabile
- BLOB



Organizzazione dei record nelle pagine

Tipicamente **una pagina contiene**

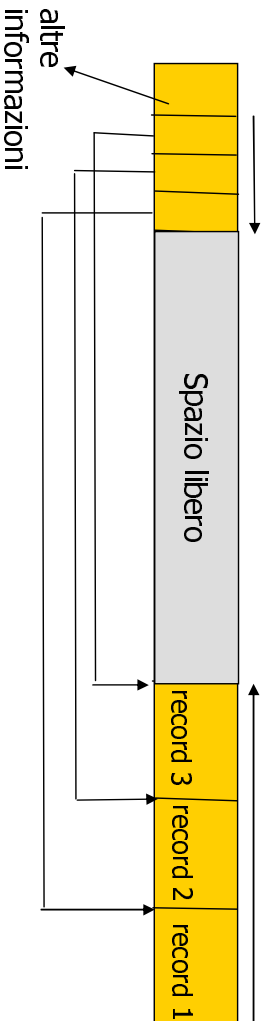
- informazioni di controllo del file system
- informazione di controllo del DBMS
 - dizionario di pagina, numero di record contenuti nella pagina, tipo di oggetto (tabella, indice,...), spazio libero, puntatore all'oggetto successivo,...
- dati
- bit di parità'



Organizzazione dei record nelle pagine II

Il **dizionario di pagina** contiene

- i puntatori ai record contenuti nella pagina
- i puntatori e i dati crescono in direzioni opposte
- altre informazioni sui record



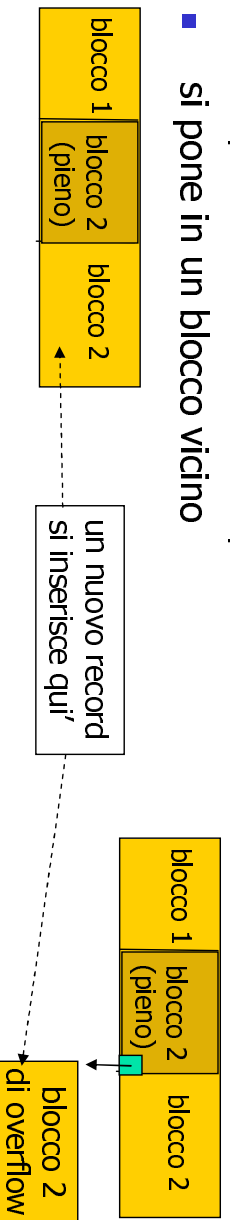
Inserimenti, cancellazioni e modifiche

Cancellare un record in un blocco

- marcare il record come eliminato
 - occorre un bit di validità per marcare l'eliminazione
 - lo spazio può essere (parzialmente rifiutato)
- rimuovere il record e riorganizzare
 - occorre fare attenzione ai puntatori

Inserire un record in una pagina, se non esiste un posto libero

- si crea un blocco di overflow
 - può essere necessario un puntatore in avanti
- si pone in un blocco vicino





Organizzazione dei record nelle pagine III

Record e pagine: **osservazioni**

- se le tuple hanno lunghezza fissa, una pagina contiene $\lfloor L_p/L_t \rfloor$ record (L_p = lunghezza pagina, L_t = lunghezza tupla)
- in generale L_p/L_t non è intero e in una pagina può rimanere spazio non occupato
- i DBMS possono anche permettere
 - che un record sia allocato su pagine differenti
 - il caso in cui $L_t > L_p$
(è possibile allocare record più grandi della dimensione della pagina)



Strutture per l'accesso ai dati

- Fino ad adesso abbiamo visto come i dati sono memorizzati, ma ... come sono ordinati e com'è possibile ritrovarli ?

Le strutture dati

- Nelle basi di dati le strutture dati usate servono ad organizzare i dati e a garantire un accesso efficiente
- Si dividono in **primarie** e **secondarie**

Le strutture dati secondarie

- non contengono i dati, ma facilitano l'accesso ai
- nei database relazionali sono gli indici



Strutture per l'accesso ai dati

Strutture **primarie**

- contengono i dati veri e propri e le strutture che ne facilitano l'accesso
- implementano le tabelle
- possono implementare anche un indice (insieme alla tabella)

Le strutture primarie

- **accesso sequenziale**
- **accesso calcolato**
- **ad albero**



Strutture ad accesso sequenziale

Nelle strutture ad accesso sequenziale:

- sono ordinate in sequenza secondo un qualche criterio
- si dividono in: **array**, **ad accesso seriale**, **ordinate**

array

- le posizioni sono individuate da indici
- utili se è possibile individuare un indice e le tuple hanno dimensione fissa
 - **condizioni che si verificano raramente**
- molto efficienti per la lettura/scrittura di una tupla



Strutture ad accesso sequenziale II

seriale

- ordinamento fisico che non dipende dal contenuto delle tuple
- gli inserimenti vengono effettuati
 - in coda (con riorganizzazioni periodiche)
 - al posto di record cancellati
- è molto diffusa per strutture primarie, associate a indici secondari
- è molto efficiente quando si vogliono leggere/modificare tutti gli elementi di una tabella



Strutture ad accesso sequenziale III

ordinate

- l'ordinamento fisico che dipende dal contenuto di un campo delle tuple
- l'accesso è efficiente
 - si usa una ricerca dicotomica
- l'inserimento di nuove tuple risulta problematico
 - lasciare degli spazi vuoti e riordinare localmente
 - inserire nuovi blocchi nel file
 - usare delle pagine di overflow
- vengono usate per implementare contemporaneamente una tabella e un indice (**ISAM** Index Sequential Access Method)



Strutture ad accesso calcolato

File hash e array

- è possibile usare un array per memorizzare un insieme di record se
 - esiste una chiave per la quale il numero dei valori possibili sia **paragonabile** al numero dei record
 - esempio: memorizzare 1000 studenti usando dei numeri di matricola che vanno da 1 a 1000
- se i possibili valori della chiave sono **molti di più** dei record, l'array spreca troppo spazio
 - esempio: il numero di matricola usa 10 caratteri

Soluzione

- si usa una funzione (hash) che **associa ad ogni chiave un indirizzo** in uno spazio (di dimensione leggermente superiore al numero di tuple da memorizzare)




Funzioni hash

Una **funzione hash**

- È una funzione h tale che $h(key)=i$:
 - key è una chiave
 - i un indirizzo di una tabella

Meccanismi per generare funzioni hash

- trasformazioni basate su cambio della base
- folding
- calcolo della radice
- ...



Collisioni

Le collisioni

- la funzione hash **non può essere iniettiva**: esiste la possibilità di collisioni
- le buone funzioni hash distribuiscono gli indirizzi in modo uniforme
 - le probabilità di collisione sono ridotte
- soluzioni
 - mettere l'elemento nella prima posizione libera
 - fare una blocco di overflow
 - funzioni hash "alternative"



Tabelle e file hash

Tabelle hash

- l'indirizzo hash identifica esattamente la locazione dove memorizzare il record
- sono usate in memoria primaria
- minimizzano il numero di record acceduti

File hash

- l'indirizzo hash identifica il blocco dove memorizzare il record: nel blocco i record sono disposti usando un altro criterio, ad esempio sequenzialmente
- sono usate in memoria secondaria
- minimizzano il numero di blocchi acceduti

Un esempio di tabella hash

- 40 record
 - tabella hash con 50 posizioni divisa in 5 blocchi:
 - 5 record in un blocco diverso da quello ricercato
- numero medio di accessi: $(40+5)/40=1,125$

nero = nessuna collisione

verde = collisione,

stesso blocco

rosso = collisione,

blocco diverso

chiave chiave mod 50

60600	0
66301	1
205751	1
205802	2
200902	2
116202	2
200604	4
66005	5
116455	5
200205	5

201159	9
200459	9
205610	10
201260	10
102360	10
205460	10
205912	12
205762	12
205617	17
205667	17

200268	18
205619	19
210519	19
200419	19
205724	24
205977	27
205478	28

200430	30
210533	33
205887	37
200138	38
102338	38

102690	40
115541	41
206092	42
205693	43
205845	45
200296	46
205796	46
206049	49

File hash

File hash

- la funzione hash produce l'indirizzo di un blocco
- le tuple del blocco sono organizzate in modo sequenziale
- diminuisce la probabilit  di overflow fra pagine diverse

nero = nessuna collisione
rosso = collisione,
blocco diverso

File hash con

- 40 record
- fattore di blocco 10
- 1 collisione
 - numero medio di accessi:
1,025

60600
66005
116455
200205
205610
201260
102360
205460
200430
102690

66301
205751
115541
200296
205796
205845

205802
200902
116202
205912
205762
205617
205667
206092
205977
205887

200268
205478
210533
200138
102338
205693

200604
201159
200419
205619
205724
206049
210519
200459

File hash II

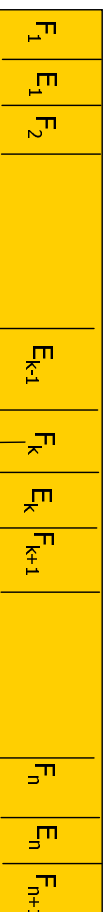
L'accesso tramite file hash

- È molto efficiente per l'accesso diretto basato su valori della chiave con condizioni di uguaglianza:
 - costo medio di poco superiore all'unità
- Non è efficiente per ricerche basate su intervalli e ricerche non basate sulla chiave
- I file hash "degenerano" se si riduce lo spazio sovrabbondante
 - funzionano con file la cui dimensione non varia molto
 - se varia occorre riordinare il file

Strutture basate su alberi di ricerca

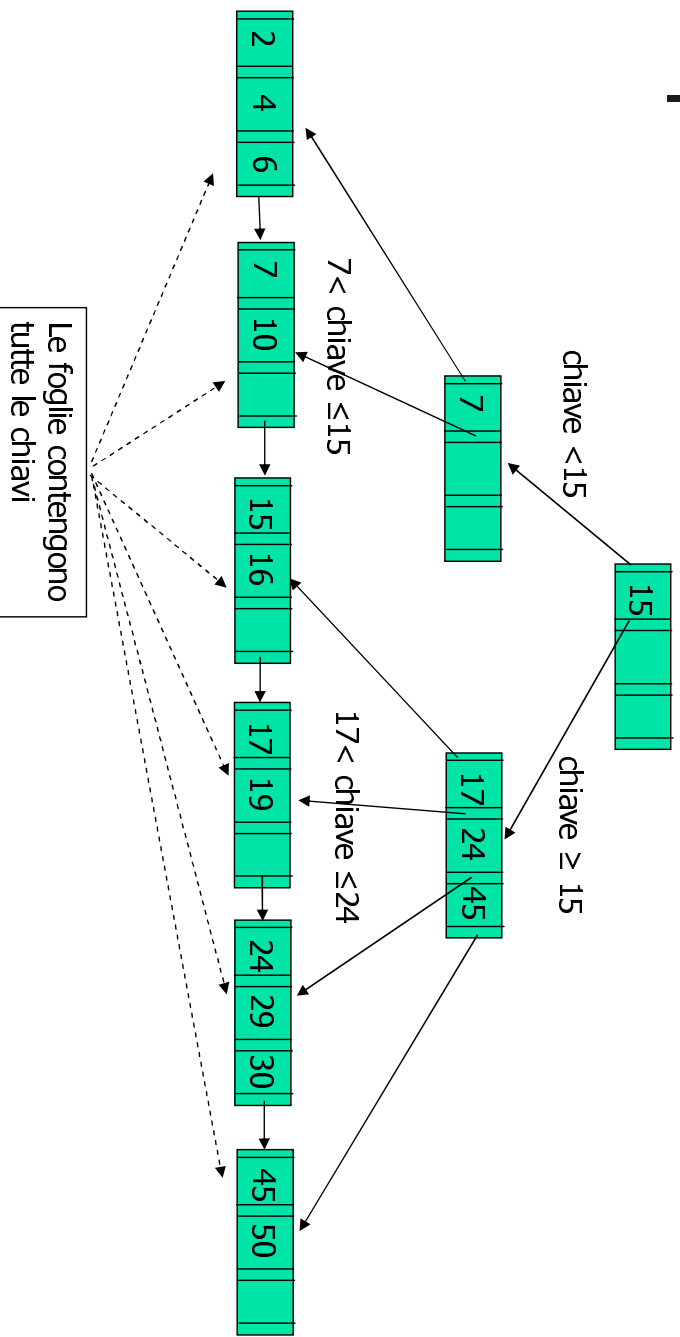
Alberi di ricerca di ordine $n+1$

- Ogni nodo ha $n+1$ figli e n etichette,
- Nell' i -esimo sottoalbero abbiamo tutte etichette maggiori della $(i-1)$ -esima etichetta e minori della i -esima
- Ogni ricerca comporta la visita di un cammino radice foglia



punta alle tuple le cui chiavi c soddisfano $E_{k-1} < c \leq E_k$

B+ Tree: un esempio



B+ tree

Un **B+ tree** è un albero di ricerca che

- viene mantenuto bilanciato
 - riempimento parziale
 - ogni nodo interno contiene almeno $\lfloor n/2 \rfloor$ chiavi
 - ogni foglia contiene almeno $\lfloor n/2 \rfloor$ chiavi
- Riorganizzazioni (locali) in caso di sbilanciamento

Inserimenti

- si fa una ricerca per trovare il punto di inserimento
- se non c'è posto nella foglia il nodo va suddiviso
- la suddivisione può propagarsi eventualmente fino alla radice

Cancellazioni e modifiche

- le eliminazioni possono portare a riduzioni di nodi
- le modifiche si trattano come eliminazioni seguite da inserimenti



B-tree e fan out

Qual è il fan out ottimo per un B-tree ?

- all'aumentare del fan out diminuiscono i livelli dell'albero
 - minore numero di nodi da scandire per accedere a un dato
- se il fan out è troppo grande, la memorizzazione di un nodo può richiedere più blocchi
 - numero di accessi alla memoria secondaria maggiore
- La soluzione migliore si ha scegliendo il fan out maggiore possibile fra quelli per i quali un nodo è memorizzabile in un blocco



Efficienza dei B-Tree

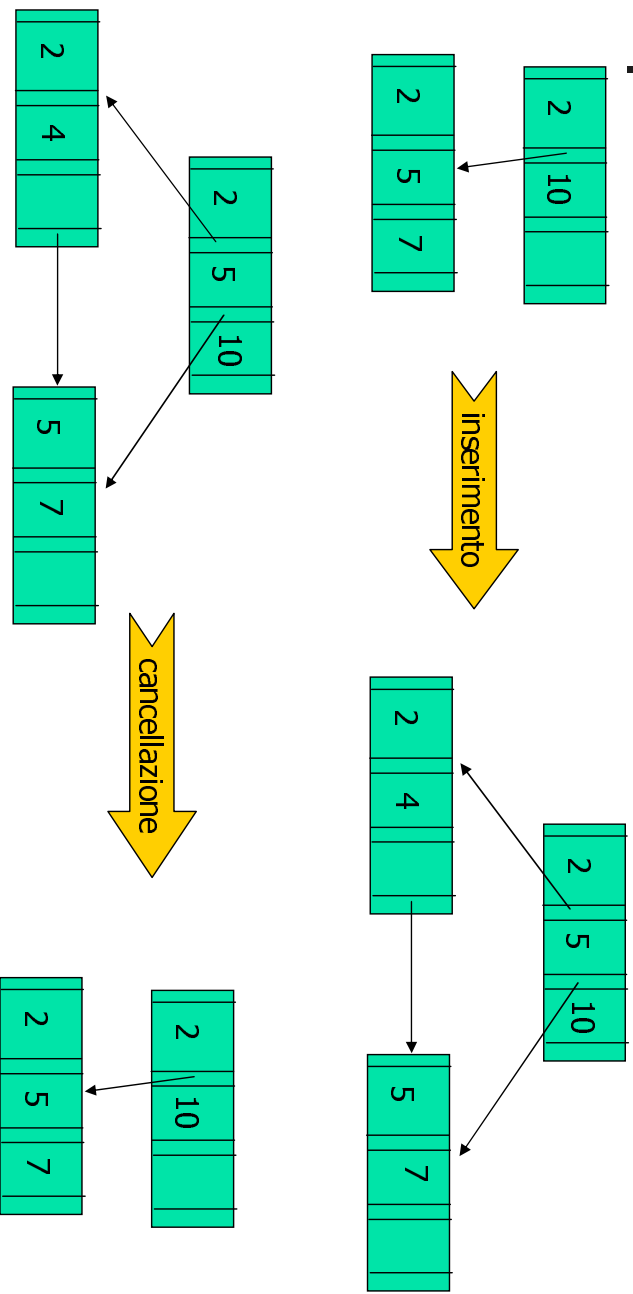
Quanto sono efficienti i B-Tree ?

- con piccole chiavi un B-Tree a tre livelli può indicizzare la maggior parte delle tabelle
- poiché la radice dell'indice può essere tenuta nel buffer, tre accessi sono sufficienti a trovare il puntatore ad un record

Esempio

- Blocchi 4KB, chiavi intere 4B, puntatori 8B, alberi di 3 livelli
- Si può calcolare il numero di chiavi per nodo:
il valore n massimo t.c. $4n + 8(n+1) < 4096$ implica $n=340$
- Supponendo che il riempimento dei nodi sia medio avremo: 255 figli per nodo
- Un B-Tree con tre livelli ha: $255^3 = 16.581.375$ foglie
- Se le foglie puntano a blocchi: si può indicizzare 68G

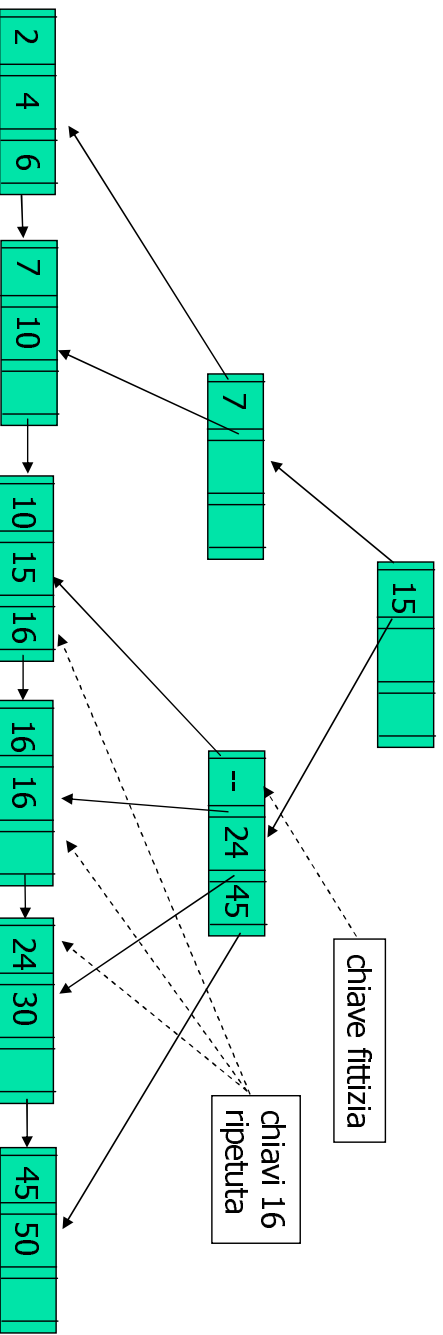
Inserimenti e cancellazioni



B+ Tree con chiavi ripetute

Cosa cambia quando ci sono **chiavi ripetute**?

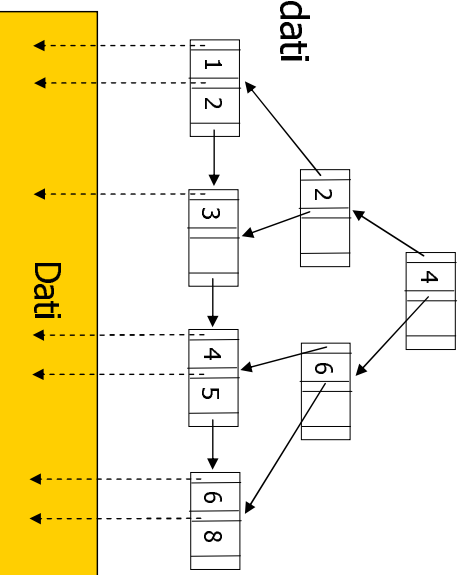
- il puntatore destro indica la foglia in cui si trova la **prima occorrenza** della chiave
- i nodi possono contenere **chiavi fittizie** (vuote)



B+ Tree e B tree

B+ tree

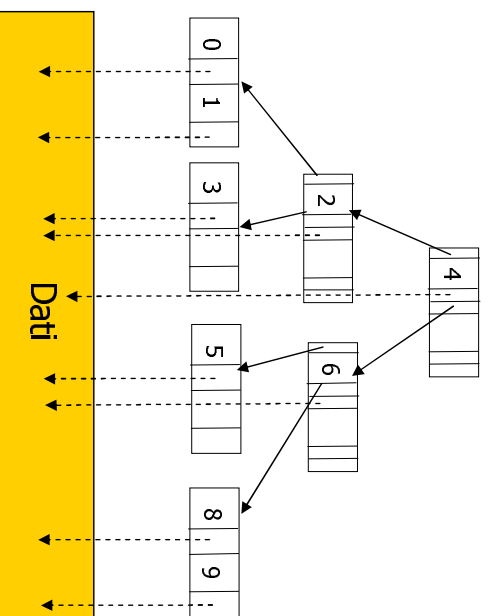
- le foglie contengono tutte le chiavi
 - alcune chiavi sono ripetute nei nodi interni
- le foglie sono collegate in una lista
- le foglie puntano ai (o contengono) dati
- ottimi per le ricerche su intervalli
- molto usati nei DBMS



B+ Tree e B tree II

B tree

- le chiavi non sono ripetute
- ogni nodo punta ai dati corrispondenti
- la struttura dati è piu' piccola rispetto ai B+ tree
- le foglie sono diverse dai nodi interni
- meno adatti alle ricerche su intervalli
- prestazioni peggiori in caso di accesso condiviso in modifica





Indici

indice primario

- su un campo sul cui ordinamento è basata la memorizzazione (es. indice generale di un libro)

```
CREATE CLUSTERED INDEX mio_indice_primario ON studenti(matricola)
```

indice secondario

- su un campo con ordinamento diverso da quello di memorizzazione (es. indice analitico di un libro)

```
CREATE INDEX mio_indice_secondario ON studenti(cognome)
```



Indici II

Osservazioni

- Ogni file può avere al più un indice primario e un numero qualunque di indici secondari
 - es. una guida turistica può avere l'indice dei luoghi e quello degli artisti
- Un file hash o ad accesso sequenziale non può avere un indice primario
- L'accesso a file con indice richiede $O(\log n)$, migliore dell'accesso sequenziale, peggiore dell'accesso a file hash

Indici III

indice denso

- contiene un record per ciascun record del file
 - gli elementi dell'indice puntano ai record
- indice sparso**
- contiene solo alcuni record del file
 - gli elementi dell'indice puntano ai blocchi

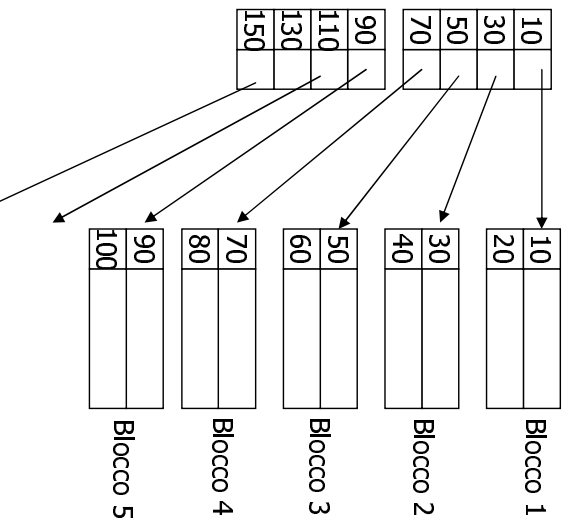
Osservazioni

- Un indice primario può essere sparso, uno secondario deve essere denso
- gli indici sparsi occupano meno spazio
- gli indici densi permettono di far alcune operazioni senza accedere ai dati (se le condizioni operano sui campi oggetto degli indici)

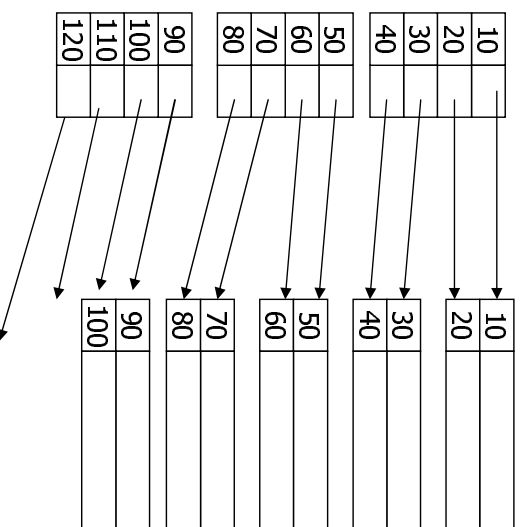
```
SELECT * FROM studenti
WHERE  cognome LIKE 'B%'
AND data_nascita >= '1/1/1980'
```

Un esempio: indici primari

Indice sparso

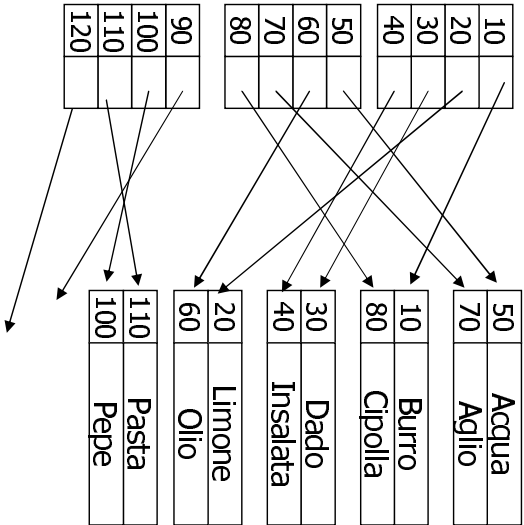


Indice denso



Un esempio: indice secondario

Indice denso



Inserimenti, cancellazioni e update

Cosa comporta la modifica dei dati ?

Azione	Indice denso	Indice sparso
Creare un blocco vuoto di overflow	Nessuna modifica	Nessuna modifica
Eliminare un blocco vuoto di overflow	Nessuna modifica	Nessuna modifica
Creare un blocco vuoto in sequenza	Nessuna modifica	Inserimento
Eliminare un blocco vuoto in sequenza	Nessuna modifica	Cancellazione
Inserire un record	Inserimento	Nessuna o modifica
Cancellare un record	Cancellazione	Nessuna o modifica
Modificare un record	Modifica	Nessuna o modifica

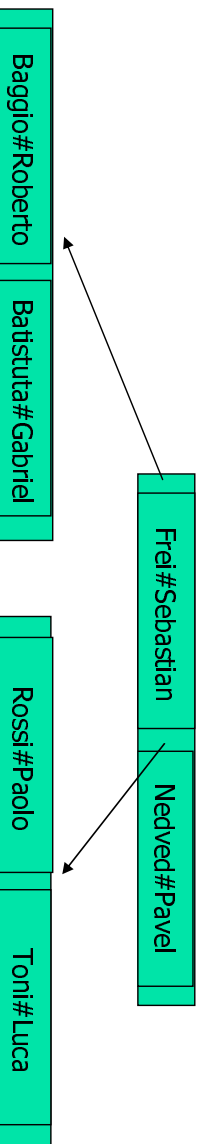
Indici: osservazioni

- Accesso diretto (per chiave) efficiente
 - sia puntuale che per intervalli
- Modifiche della chiave, inserimenti, eliminazioni possono richiedere una riorganizzazione dell'indice
 - tecniche per alleviare i problemi:
 - file o blocchi di overflow
 - marcatura per le eliminazioni
 - riempimento parziale
 - blocchi collegati (non contigui)
 - riorganizzazioni periodiche

Multi-indici e strutture dati per applicazioni particolari

- Gli indici su più campi possono essere facilmente realizzati con alberi ordinati rispetto alla concatenazione

```
CREATE INDEX indice1 ON studenti(cognome, nome)
```



- Anche file hash con chiavi composte possono essere costruiti nello stesso modo
- In alcuni casi, però, conviene usare strutture dati multidimensionali ad hoc per il problema considerato

Esempio: i documenti di testo

In information retrieval, un **documento testuale** è rappresentabile con

- un record con un numero di campi uguale alla **dimensione del vocabolario**
- l'i-esimo campo è **True** o **False** a seconda se l'i-esima parola del vocabolario sia presente o meno nel documento

Questo è un documento che parla di Linux



a	abbaiare	abbaiano	...	documento	...	parla	...	linux	...
False	False		...	True	...	True	...	True	...

L'uso di **indici tradizionali** in **questo caso non è indicato perché**

- Occorrerebbe un indice per ogni campo (parola)
- Ogni campo accetta solo due valori: un indice su un campo si ridurrebbe ad un albero con due soli rami
- C'è un numero grande e variabile di campi

Esempio: i documenti di testo II

Intuitivamente

- si crea un unico indice, detto **indice inverso**, che combina gli indici su ogni campo
- l'indice punta solo a documenti che contengono le parole (True) e non a quelli che non le contengono (False)

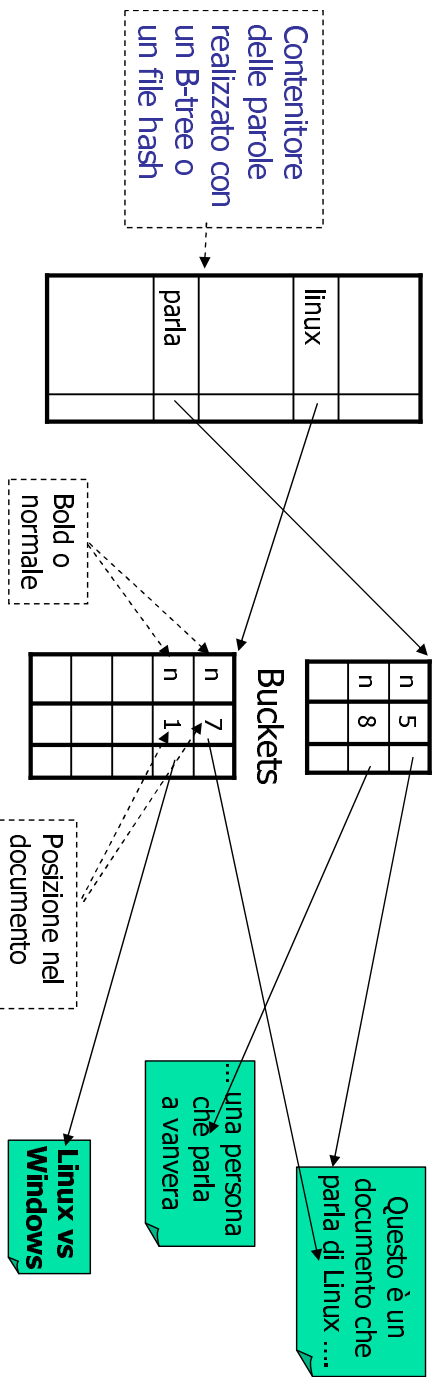
In pratica

- l'indice contiene tutte le parole presenti nell'archivio e può essere organizzato usando una tecnica tradizionale (B-tree, file hash)
- l'indice punta a dei **bucket** che contengono una lista delle occorrenze di ciascuna parola eventualmente con l'aggiunta di altre informazioni sull'occorrenza

Esempio: i documenti di testo III

Gli indici inversi

- usano dei buckets (indicizzazione indiretta) per gestire le ripetizioni
- possono contenere ulteriori informazioni sulle parole
 - posizione nel documento dove si trova parola
 - caratteristiche del formato della parola (bold, nel titolo, carattere normale, ..)



Altre applicazioni

Geographic Information System (GIS)

- Memorizzano dati rappresentabili in due o tre dimensioni
- Servono a gestire mappe geografiche, ma anche disegni di circuiti, architettura,
- Permettono di registrare punti, rettangoli, cerchi e più in generale ponti, strade, etc

Data Cube

- Memorizzano informazione multi-dimensionale per applicazioni di supporto alle decisioni: ad esempio, una catena potrebbe memorizzare per ogni acquisto data, prezzo, negozio, oggetto acquistato, guadagno ...
- Permettono di visualizzare i dati come cubi dove ogni dimensione corrisponde ad un attributo

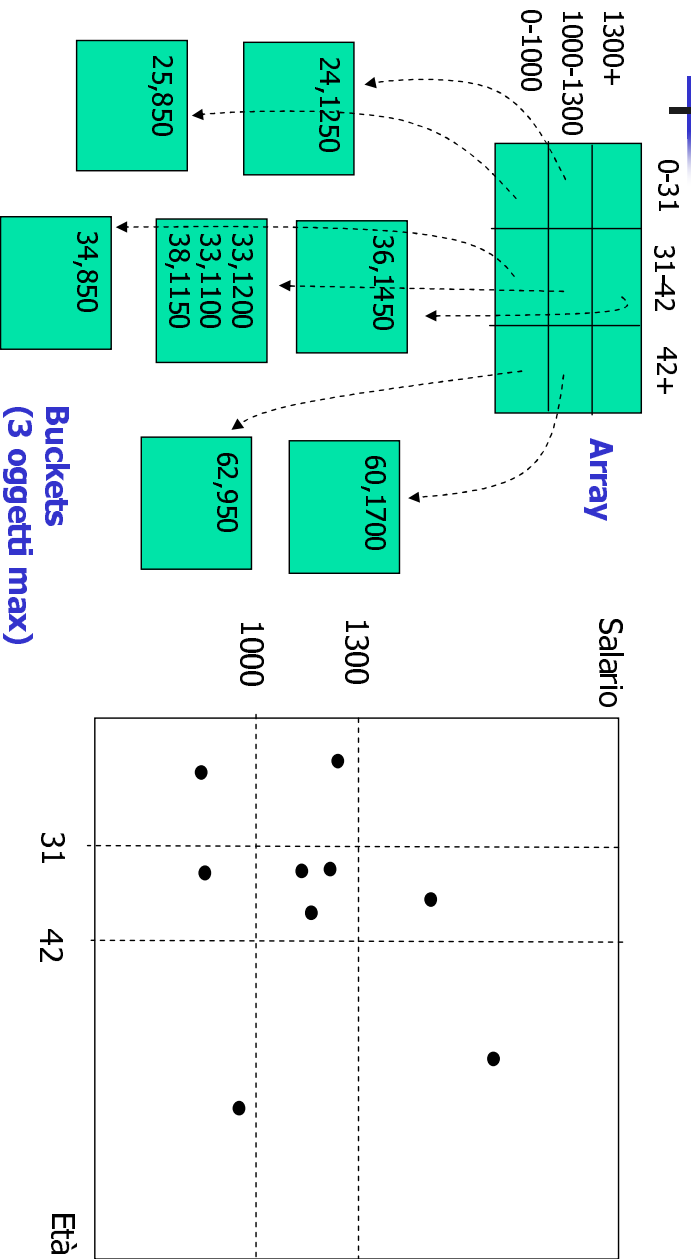
Altre applicazioni II

- GIS e Data Cube implementano operazioni che possono essere rese più efficienti utilizzando indici ad hoc
- **Aggregazioni**
 - “Quante magliette azzurre sono state vendute in nei negozi toscani nel primo trimestre del 2005 ?”
- **Interrogazioni con match parziale**
 - “Quali sono le università a nord di Siena?” ,
 - “Quali sono le università ad est di Siena?” ,
- **Interrogazioni su intervalli**
 - “Quali sono le città fra il meridiano xx e quello yy?” ,
 - “Quali sono i fiumi fra il meridiano xx e quello yy?” ,
- **Ricerca dei più vicini**
 - “Qual è l'autostrada più vicina a Siena?”
 - “Qual è la regione non confinante più vicina alla toscana?”
- **Interrogazioni “dove mi trovo?”**
 - “In quale contrada mi trovo?”

Strutture dati per dati multi-dimensionali hash-like

- Si partiziona lo spazio in regioni.
- Per ogni regione si realizza un bucket dove si inseriscono tutti gli oggetti in essa contenuta.
- L'indirizzo del bucket viene calcolato applicando una funzione alla chiave.
- Esempi di tecniche di questo tipo:
 - **Grid Files**
 - Le regioni sono definite dal linee orizzontali e linee verticali
 - Una matrice contiene i puntatori ai bucket
 - **Partitioned Hashing**
 - si applica una funzione hash h ad ogni dimensione v_i
 - la chiave hash globale $h(v_1)..h(v_n)$ che definisce quale bucket usare è data dalla concatenazione delle singole chiavi hash $h(v_i)$

Grid Files



Grid Files

Inserimenti e cancellazioni

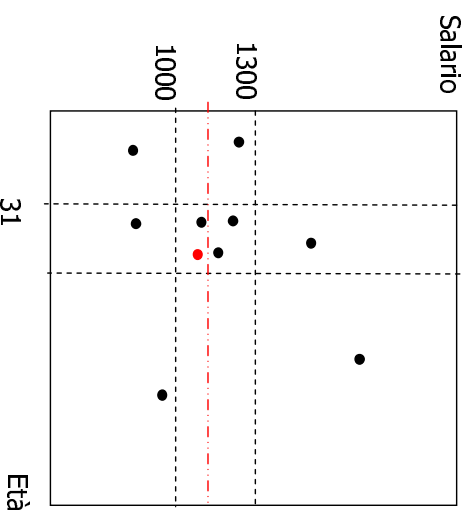
- possono richiedere la divisione o la fusione delle regioni e dei bucket

Interrogazioni su intervalli e "dove mi trovo?"

- vengono eseguite considerando le regioni (i bucket) individuati dagli intervalli

Ricerca dei più vicini ad un oggetto o

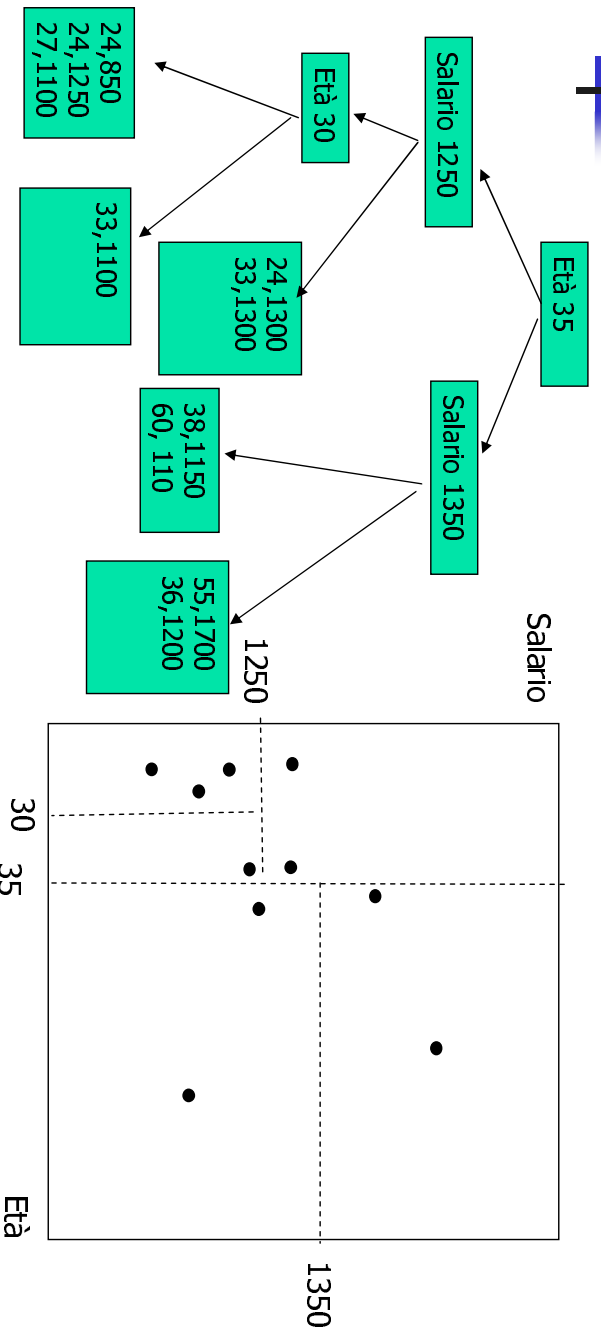
- Si fissa un valore d e si cercano in tutte le regioni che possono contenere oggetti più vicini di d da o
- Se non si trova nessun oggetto si riesegue l'interrogazione con un valore più grand di d



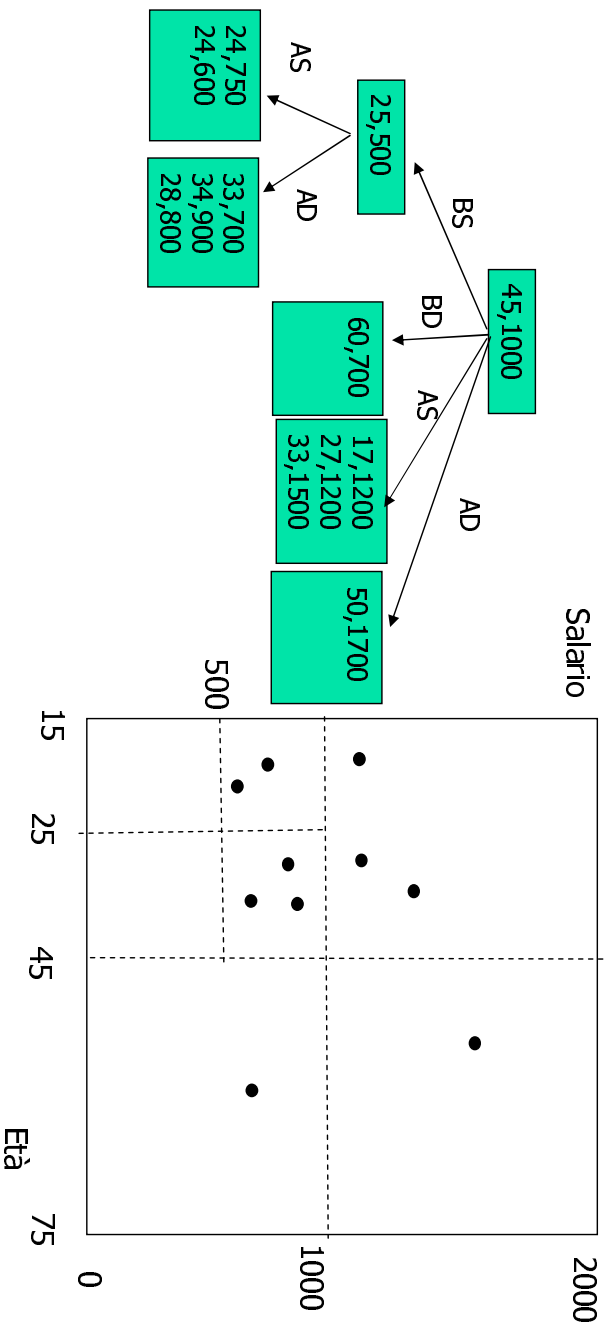
Strutture dati per dati multi-dimensionali basate su alberi

- Si partiziona lo spazio in regioni e sottoregioni
- Si costruisce un albero in cui ogni nodo rappresenta una regione
- La radice rappresenta la regione di tutto lo spazio ammissibile, mentre i figli di ciascun un nodo rappresentano sottoregioni di quella del nodo
- Le foglie dell'albero sono i bucket in cui sono contenuti gli oggetti
- Esempi di tecniche di questo tipo:
 - **KD-Tree**
 - In ogni nodo la regione viene divisa rispetto ad un valore ***a*** di un attributo: in un ramo gli oggetti maggiori di ***a***, nell'altro quelli minori
 - L'attributo scelto è lo stesso per tutti i nodi di un livello
 - **Quad-Tree**
 - Ogni regione viene suddivisa in quattro sottoregioni uguali corrispondenti a quattro quadranti
 - **R-Tree**
 - Le regioni hanno tutte forma predefinita: ad. es. un rettangolo
 - Le sottoregioni sono scelte possono avere delle intersezioni e non coprire tutta la regione padre

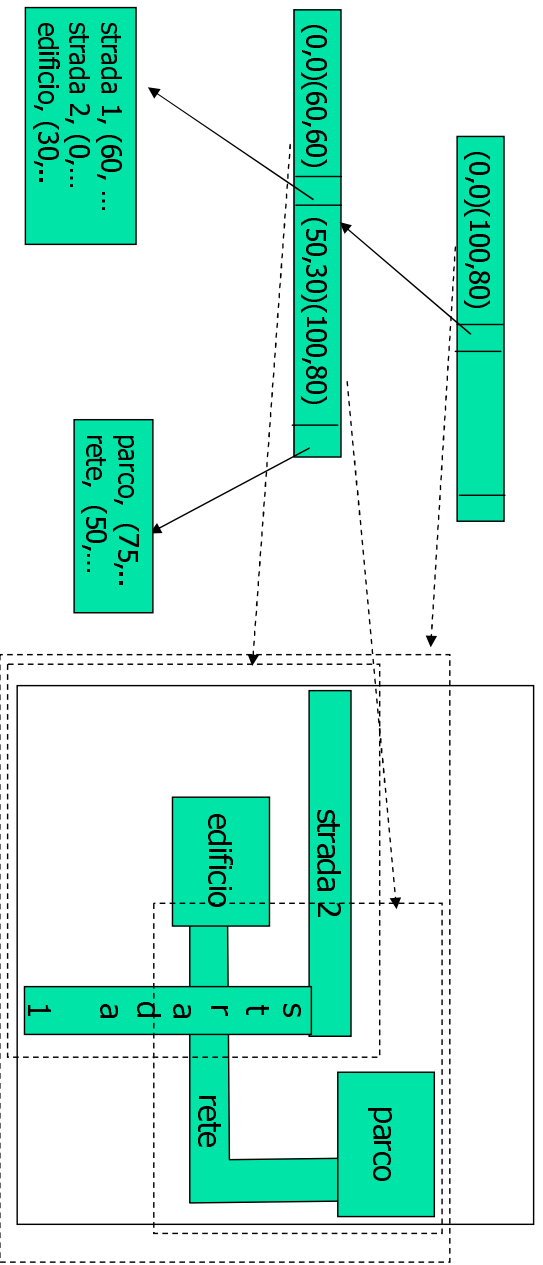
KD-Tree



Quad-Tree



R-Tree





Strutture dati per dati multi-dimensionali basate su bitmap

- Per ogni attributo si considerano tutti i valori V_{j1}, \dots, V_m che compaiono nell'archivio
- Un **indice bitmap** su un attributo consiste in un vettore di ***m*** bit per ciascun record
- Ogni vettore è costituito da tutti 0 e un solo 1 in corrispondenza del valore del record
- Un indice multi-dimensionale si ottiene concatenando gli indici bitmap degli attributi coinvolti
- Con gli indici bitmap si implementano facilmente attraverso operazioni di OR e AND su bit, le operazioni di intervallo incluse in un'interrogazione



Indici bitmap: un esempio

Esempio

- Dati i record (Età, salario): (30,1000),(30,1300), (40,1000), (50,1300)
- con 100 si rappresenta 30, con 010 si rappresenta 40, con 001 si rappresenta 40
- con 10 si rappresenta 1000, con 01 si rappresenta 1300

Dati	Indice bitmap su Et�	Indice bitmap su salario	Indice bitmap su entrambi
(30,1000)	100	10	10010
(30,1300)	100	01	10001
(40,1000)	010	10	01010
(50,1300)	001	01	00101

.... per cercare gli impiegati di et  compresa fra 30 e 40, si cercano i bitmap dove il primo o il secondo bit sono 1

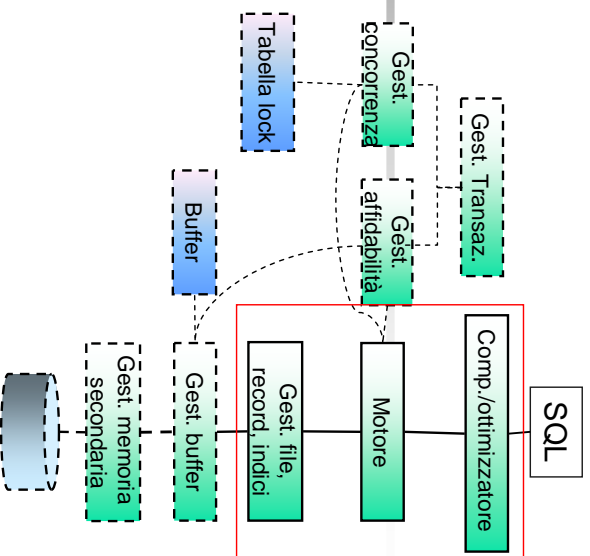
Indici bitmap

- Per diminuire l'occupazione di memoria gli indici bitmap devono essere compressi
 - Un metodo possibile è il seguente
 - Si codificano solo le **lunghezze delle sequenze di 0**: si suppone che ci sia un 1 fra ogni sequenza
 - Ogni lunghezza è codificata dal valore **v** in binario preceduto da **uno 0 e tanti 1** (meno uno) quanti sono i bit di **v**
 - Il metodo è **efficiente quando ci sono pochissimi 1**: ad esempio, quando nella codifica del testo dove un attributo indica al presenza di una parola
- | | |
|----------------------------|--|
| Codifica della lunghezza 5 | Codifica della bitmap 0001000 |
| 1 10 1 01 | 1 0 1 1 0 1 1 |
| Codifica della lunghezza 3 | Codifica della bitmap 0100000 |
| 1 0 1 1 | 0 1 1 1 0 101 |
| Codifica della lunghezza 1 | Codifica della bitmap 0111000 |
| 0 1 | 0 1 0 0 0 0 1 0 1 1 |
| Codifica della lunghezza 0 | |
| 0 0 | |

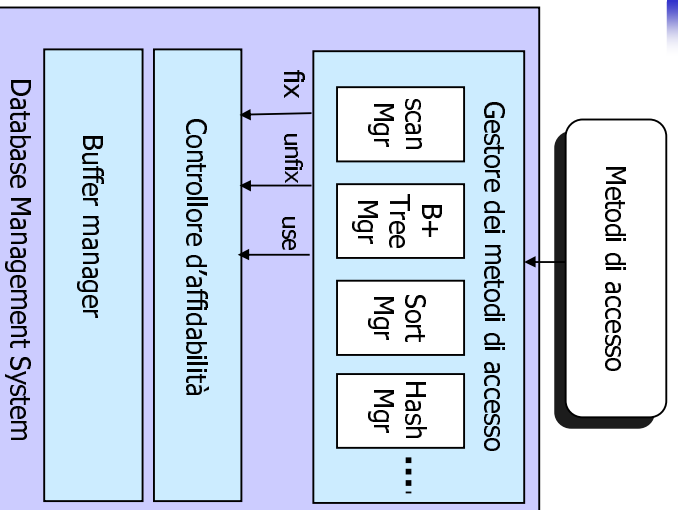
Osservazioni

- Per alcune delle strutture ad albero presentate, esistono delle estensioni ad alberi n-ari: l'obiettivo è quello di memorizzare un nodo esattamente in blocco come per i B-Tree
- In alternativa, in un blocco si memorizzano un nodo e una parte dei suoi discendenti: questo permette di rendere più efficiente la ricerca
- Per una dimostrazione delle strutture dati presentate si veda. ad esempio,
<http://donar.umiacs.umd.edu/quadtrees/>

Esecuzione delle interrogazioni

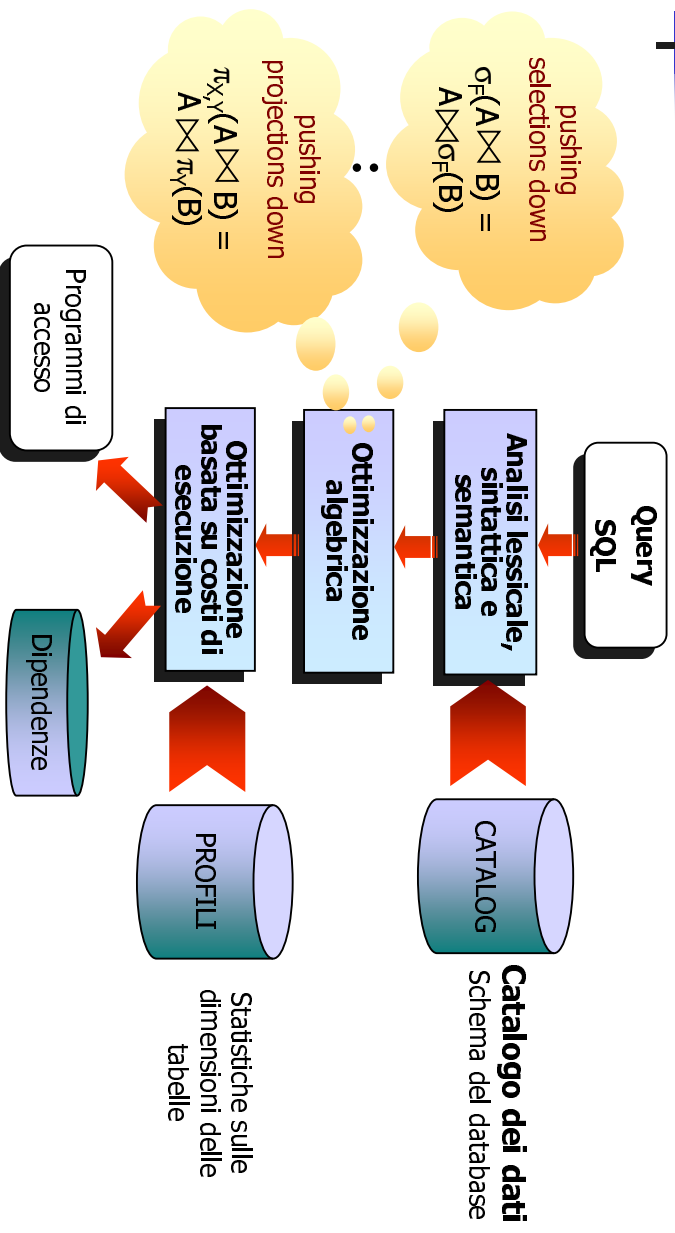


Strutture fisiche di accesso



- Riguardano l'organizzazione dei dati per rendere efficienti le operazioni di ricerca e modifica
- Si possono definire indici
- I metodi di accesso gestiscono una particolare organizzazione fisica
- Il metodo di accesso individua i blocchi fisici che devono essere caricati in memoria dal buffer manager

Ottimizzazione delle interrogazioni



Compilazione delle query

- I programmi di accesso sono in formato "oggetto"
- **compile-and-store**
 - l'interrogazione viene compilata una sola volta
 - il codice oggetto viene memorizzato insieme alle dipendenze dalle versioni di tabelle e indici
 - il codice oggetto viene invalidato se la struttura della base di dati cambia significativamente per l'interrogazione (es. aggiunta di un indice)
- **compile-and-go**
 - l'interrogazione viene compilata ed eseguita, ma non è memorizzata

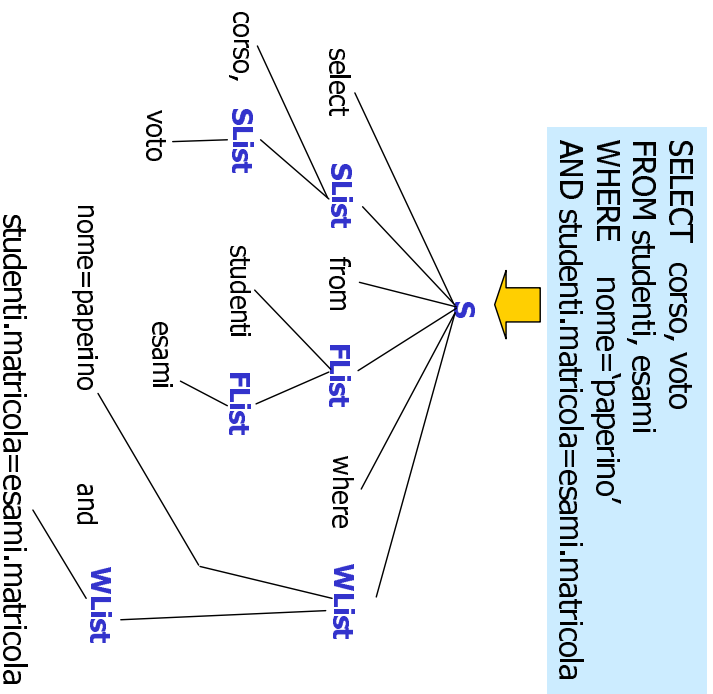
Analisi lessicale, sintattica e semantica

Analisi lessicale e sintattica

- un parser legge la query in SQL e produce il rispettivo albero di analisi
- il parser è costruito a partire dalla grammatica dell'SQL con le tecniche usate anche per gli altri linguaggi

Analisi semantica:

- si controlla l'esistenza delle tabelle e attributi indicati nell'interrogazione
- si associa ad ogni nome il relativo oggetto
- si controlla che i tipi dei dati coinvolti nelle operazioni sia corretto



Traduzione dell'interrogazione in algebra relazionale

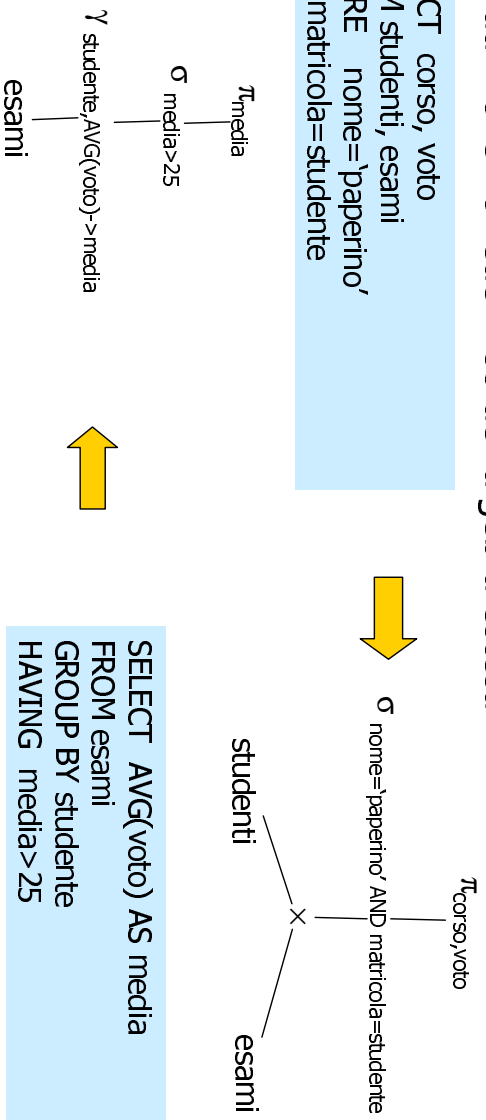
Traduzione in algebra relazionale

- l'albero sintattico viene tradotto in un altro albero che rappresenta l'interrogazione in **algebra relazionale estesa**
 - le foglie contengono tabelle
 - i nodi interni operatori relazionali
- operatori dell'algebra
 - unione, intersezione, differenza
 - selezione (ad es. $\sigma_{\text{conome=paperino}}(\text{STUDENTI})$)
 - proiezione (ad es. $\pi_{\text{matricola}}(\text{STUDENTI})$)
 - prodotto e join ($\times, \triangleright \triangleleft$)
 - aggregazione (ad es. $\gamma_{\text{matricola, AVG(voto)} \rightarrow \text{media}}(\text{ESAMI})$)
 - ordinamento (ad es. $\tau_{\text{nome}}(\text{STUDENTI})$)

Traduzione dell'interrogazione in algebra relazionale II

- nel caso di una interrogazione select-from-where la trasformazione è semplice
- se l'interrogazione contiene operatori di raggruppamento o ordinamento, la traduzione richiede l'uso dell'algebra estesa

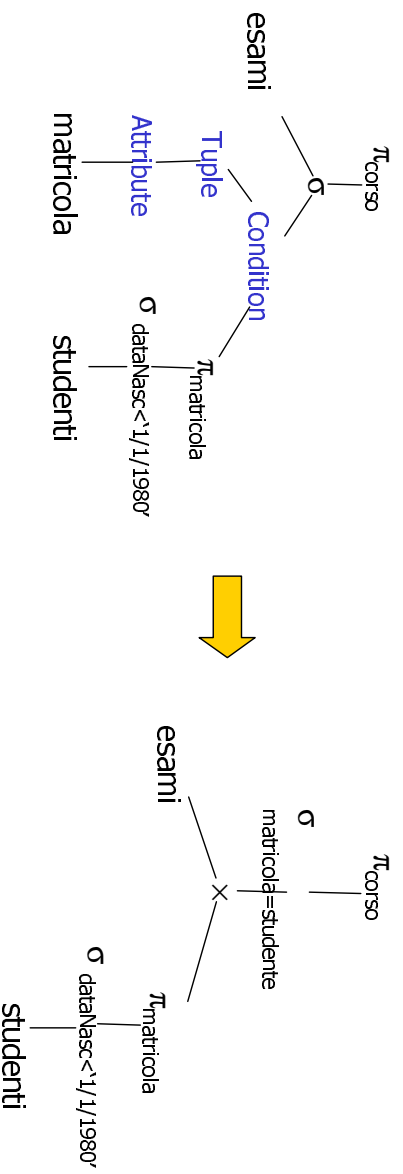
```
SELECT corso, voto
FROM studenti, esami
WHERE nome='paperino'
AND matricola=studente
```



Traduzione dell'interrogazione in algebra relazionale III

- nel caso di una interrogazione che contenga una sottointerrogazione
 - prima, si trasforma parzialmente l'interrogazione in un albero con un nodo speciale che rappresenta la sottointerrogazione
 - poi, si tenta di riscrivere la sottointerrogazione in una espressione equivalente

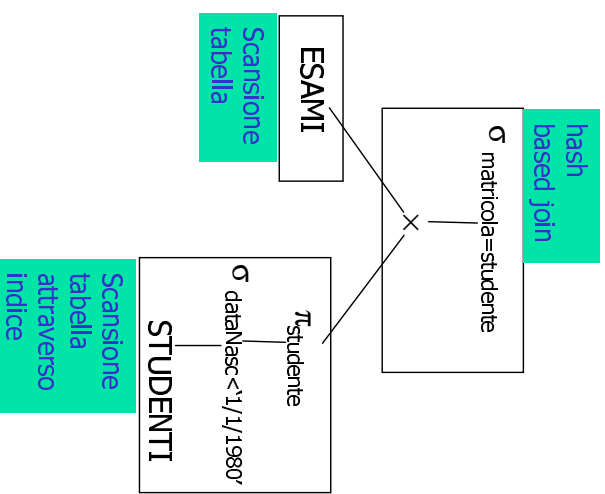
```
SELECT corso FROM esami
WHERE studente IN (SELECT matricola FROM studenti WHERE dataNasc < '1/1/1980')
```



Query plan

Query plan

- è una **rappresentazione interna** della query
- query plan **logico**
 - è un albero rappresentante un'espressione relazionale
 - suggerisce le operazioni logiche da eseguire e l'ordine in cui devono essere eseguite
- query plan **fisico**
 - è un albero indicante le operazioni fisiche necessarie ad implementare l'interrogazione
 - ogni nodo implementa un insieme di operazioni descritte dal query plan logico



Query plan II

La trasformazione da query plan logico a query plan fisico

- **ottimizzare il query plan logico** (ottimizzazione algebrica)
 - si cercano, fra quelle algebricamente equivalenti, le espressioni piu' efficienti
- **implementare gli operatori dell'algebra relazione**
 - trasformare gli operatori in operazioni da richiedere al gestore dei metodi d'accesso
- **ottimizzare il query plan fisico** (ottimizzazione basata sui costi di esecuzione previsti)
 - scegliere fra le possibili implementazioni quella piu' efficiente sulla base dello stato attuale del database

Ottimizzazione algebrica

Il primo passo dell'ottimizzazione

- si riscrive l'espressione relazionale in nuove espressioni
 - equivalenti
 - piu' efficienti
- in questa fase **non si usa informazione** sullo stato attuale del database
- la riscrittura si basa su formule di equivalenza

Idee guida

- minimizzare i numero di record nelle relazioni intermedie
- minimizzare la dimensione delle relazioni intermedie

Ottimizzazione algebrica II

Pushing selections

- consiste nel spingere il piu' possibile **verso le foglie** (anticipare) le selezioni
- minimizza il numero di record trattati
- alcune equivalenze:
 - $\sigma_C(R-S) = \sigma_C(R) - \sigma_C(S)$ (puo' convenire piu' di $\sigma_C(R-S) = \sigma_C(R) - S$)
 - $\sigma_C(R \bowtie_{\triangleright \triangleleft D} S) = \sigma_C(R) \bowtie_{\triangleright \triangleleft D} \sigma_C(S)$
 - $\sigma_C(R \bowtie_{\triangleright \triangleleft D} S) = \sigma_C(R) \bowtie_{\triangleright \triangleleft D} S$ (se C non contiene attributi di S)
 -

$\pi_{\text{corso, voto}}(\sigma_{\text{nome}=\text{'paperino'}}(\text{ESAMI} \bowtie_{\text{matricola}=\text{studente}} \text{STUDENTI}))$



$\pi_{\text{corso, voto}}(\text{ESAMI} \bowtie_{\text{matricola}=\text{studente}} (\sigma_{\text{nome}=\text{'paperino'}}(\text{STUDENTI})))$

Ottimizzazione algebrica III

Pushing projections

- consiste nel spingere il piu' possibile verso le foglie (anticipare) le proiezioni
- piu' in generale le proiezioni possono essere aggiunte ovunque, purché tolgano solo **attributi che non verranno piu' usati**
- alcune equivalenze
 - $\pi_L(\sigma_C(R)) = \pi_L(\sigma_C(\pi_M(R)))$
dove M sono gli attributi di R che compaiono in C o in L
 - $\pi_L(R \bowtie_D S) = \pi_L(\pi_M(R) \bowtie_D \pi_N(S))$
dove M (N) sono gli attributi di R (S) che compaiono in D o in L

$\pi_{\text{corso, voto}}(\sigma_{\text{nome}=\text{'paperino'}}(\text{ESAMI} \bowtie_{\text{matricola}=\text{studente}} \text{STUDENTI}))$



$\pi_{\text{corso, voto}}(\text{ESAMI} \bowtie_{\text{matricola}=\text{studente}} (\sigma_{\text{nome}=\text{'paperino'}} (\pi_{\text{nome, matricola}}(\text{STUDENTI}))))$

Realizzazione delle operazioni fisiche

Le operazioni implementate in RDBMS tipici sono

- scansioni sequenziali
- join
- ordinamenti
- accessi tramite indice

Un'altra classificazione

- metodi one-pass che richiedono **una sola lettura** dei dati
 - di solito funzionano se uno degli operandi sta tutto in memoria
- metodi **two-pass** che richiedono **due letture** dei dati
 - ad esempio, il two phase multiway merge-sort
- metodi che richiedono **piu' letture** dei dati



Scansioni (scan)

- Si accede in sequenza alle tuple
 - open - si apre la scansione
 - next - si avanza il puntatore alla tupla successiva
 - read/modify/delete - legge/modifica/cancella la tupla corrente
 - insert - inserisce una nuova tupla nella posizione corrente
 - close - chiude la scansione
- Durante lo scan si possono applicare delle operazioni
 - proiezione, selezione su un predicato semplice, ordinamento



Accesso tramite indici (index scan)

- Gli indici sono spesso realizzati con alberi
- Favoriscono l'accesso associativo per interrogazioni che comprendono predicati tipo $A_i = v$ oppure $v_1 \leq A_i \leq v_2$ (predicati valutabili con l'indice)
- $C_1 \wedge C_2$ dove entrambe le condizioni sono valutabili con indici (riguardano le chiavi), si sceglie la più selettiva fra le due per l'accesso tramite indice. Il secondo predicato viene valutato sulle pagine caricate nel buffer. Se gli indici sono densi si possono valutare entrambe le condizioni sugli indici e poi unire i risultati.
- $C_1 \vee C_2$ se una delle due non è valutabile con indice, occorre fare una scansione completa
- $C_1 \vee C_2$ se sono entrambe valutabili si possono usare gli indici ma non è detto che sia conveniente (se c'è molta sovrapposizione)

Accesso tramite indici II

Definiamo

- $B(R)$ – numero di blocchi che costituiscono la relazione R
- $T(R)$ – numero di record che costituiscono la relazione R
- $V(R,A)$ – numero di valori diversi per l'attributo A di R
- M dimensione in blocchi della memoria disponibile

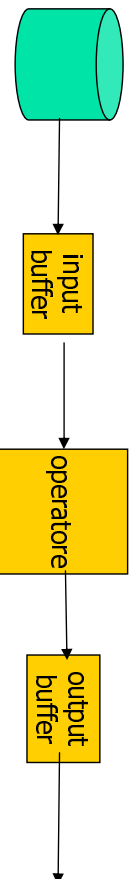
Costo di realizzazione della selezione $\sigma_{A=v}(R)$

- **clustered index su $R.A$ (tipico per indici primari)**
se i record che hanno valore uguale per A sono memorizzati il più vicino possibile nell'indice
 - circa $B(R)/V(R,A)$
- **nonclustered index su $R.A$ (tipico indici secondari densi)**
se i record che hanno valore uguale sono memorizzati in blocchi diversi
 - circa $T(R)/V(R,A)$
- **senza indice**
 - circa $B(R)$

Metodi one-pass: esempi

Selezione, proiezione, raggruppamento e altri operatori unari

- funzionamento
 - si leggono i dati dalla memoria secondaria nel buffer
 - si applica l'operatore
 - si scrive il risultato nel buffer di output
- costo (senza la scrittura del risultato su disco!)
 - se i dati sono **clustered**, B (numero dei blocchi) accessi alla memoria secondaria
 - se i dati sono **nonclustered**, T (numero di record) accessi alla memoria secondaria
 - il raggruppamento funziona solo se $B < M$



Metodi one-pass: esempi II

Join

- funzionamento
 - si carica la prima relazione R dalla memoria secondaria alla memoria primaria (ad. es. tabella hash)
 - si costruisce una struttura di accesso ad R
 - si legge la seconda relazione S un blocco alla volta
 - per ogni record di S si controlla se esiste uno o più record in R che soddisfano il join e si scrive il record risultato in un buffer di uscita
- costo
 - se i dati sono **clustered** $B(R)+B(S)$
 - funziona solo se $B(R) < M$

Metodi two pass

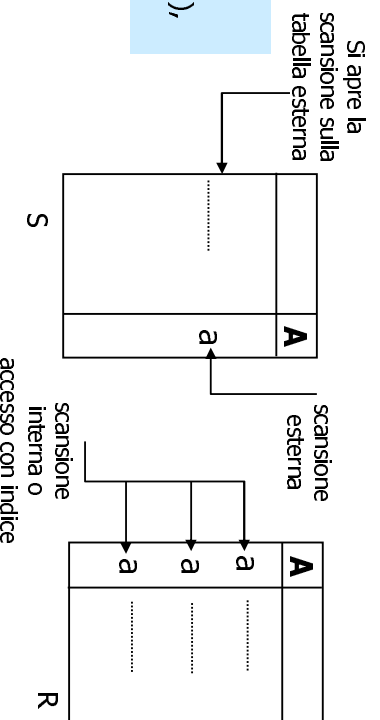
Cosa succede se le relazioni non entrano in memoria ?

Nested-loop Join

- si sceglie la relazione con un **numero minore di record (S)**
- si scandisce S e per ogni record di S si lancia una scansione su R

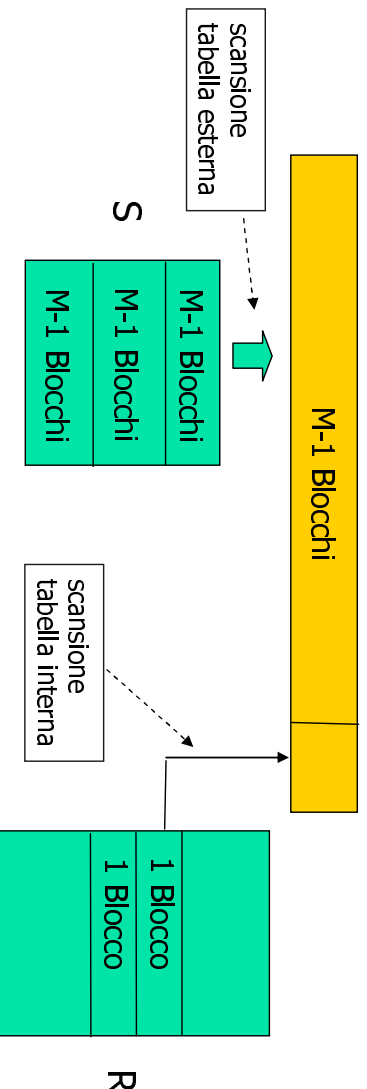
Servono $B(S)+T(S)*B(R)$ letture

```
FOR EACH tupla s in S DO
  FOR EACH tupla r in R DO
    IF (r and s soddisfano la condizione di join),
      THEN costruisci la tupla di unione
```



Nested-loop join ottimizzato

- un miglioramento si ottiene di usare un indice per R
 - costo con R e S clustered: circa $B(S) + T(S) * (B(R) / V(R, A))$
- oppure caricando nei buffer M-1 record di S e confrontandoli tutti con i record di R recuperati dalla seconda scansione
 - costo con R e S clustered: circa $(B(S) / (M-1)) * (M-1 + B(R)) = B(S) + B(R) * B(S) / (M-1)$
 - **Se non esiste un indice, si usa sempre questa strategia che è sempre migliore di quella della slide precedente!**



Merge scan join

Funzionamento

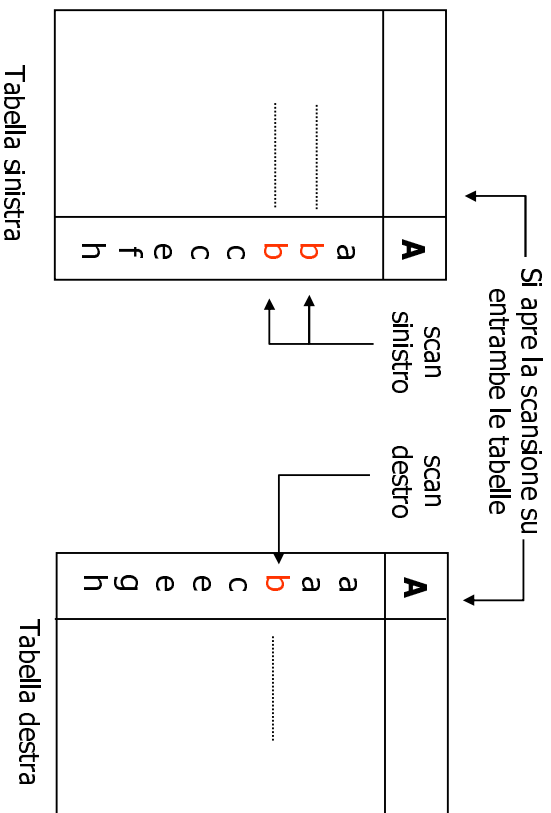
- si ordinano le due relazioni R e S
- si fondono le due relazioni scandendole contemporaneamente

Osservazioni

- il costo di questo metodo è $5 (B(R) + B(S))$
 - $4 (B(R) + B(S))$ è il costo dell'ordinamento, $(B(R) + B(S))$ il costo del merge
- un miglioramento si ottiene fondendo la seconda fase dell'ordinamento con il "merge" $3(B(R) + B(S))$
 - **questa soluzione è sempre preferibile alla precedente**
- se sono definiti degli indici, l'ordinamento non è più necessario e il costo diviene $(B(R) + B(S))$
- nested-loop join è più efficiente solo se una delle due relazioni è particolarmente piccola
- L'algoritmo richiede $M^2 > (\max(B(R), B(S)))$

Merge scan join II

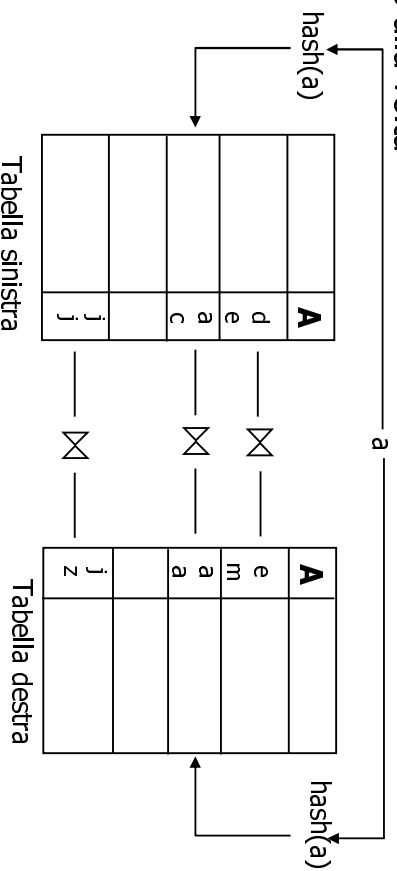
- Le tabelle devono essere ordinate in base agli attributi di join



Hash based join

Funzionamento

- le relazioni R e S sono **riorganizzate** attraverso una **funzione hash** che indica in quale bucket (fra M-1) ogni record deve essere memorizzato
 - in questo modo record con lo **stesso campo** finiscono nello **stesso bucket**
- si carica un bucket di S e il corrispondente di R e si uniscono i record
 - la memoria deve contenere uno dei bucket, l'altro può essere caricato un blocco alla volta



Hash based join

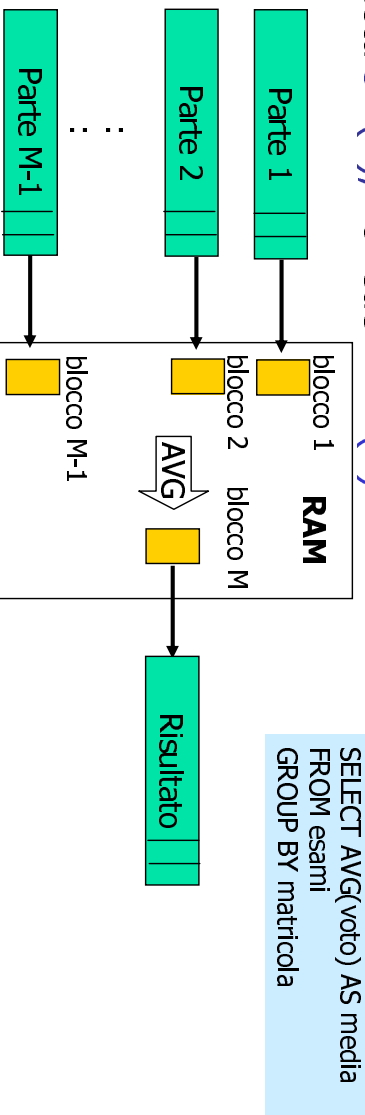
Osservazioni

- il costo di questo metodo è $3 (B(R) + B(S))$
 - $2 (B(R) + B(S))$ è il costo della prima fase
 - $(B(R) + B(S))$ il costo della seconda fase
- si richiede che la memoria contenga i bucket di S
 - implica $M^2 > \min(B(R), B(S))$
- l'hash based join può essere leggermente più veloce del merge scan join se non sono presenti indici
- l'hash based join può trattare relazioni leggermente più grandi
- il merge scan join ha il vantaggio di lasciare ordinati i dati

Altri esempi di algoritmi two pass

Raggruppamento basato su ordinamento

- si suddivide la relazione R in M-1 sottoparti R_1, \dots, R_{M-1}
- si ordinano R_1, \dots, R_{M-1} in base alla chiave del group by
- si caricano R_1, \dots, R_{M-1} in M-1 blocchi
- si scelgono i record con la stessa chiave valutando contemporaneamente l'espressione
- Costa $3 B(R)$, richiede $M^2 > B(R)$



Altri esempi di algoritmi two pass II

Raggruppamento basato su Hashing

- si suddivide la relazione R in $M-1$ buckets usando la funzione hash sulle chiavi di raggruppamento
- ogni bucket viene caricato in memoria, si scelgono i record con la stessa chiave valutando contemporaneamente l'espressione
- occorrono $3B(R)$ accessi alla memoria secondaria
- deve essere verificato $M^2 > B(R)$

Algoritmi che richiedono piu' di tre passi

Algoritmi Multipass

- servono ad elaborare relazioni troppo grandi per essere elaborate in due passi
- esistono estensioni degli algoritmi basati sull'ordinamento e di quello basati su hashing

Basati su ordinamento

- i dati vengono **ricorsivamente** divisi in $M-1$ parti R_1, \dots, R_n uguali ($n > M$) fino quando la memoria non li contiene
- le parti R_1, \dots, R_n sono riunite insieme valutando contemporaneamente la funzione voluta

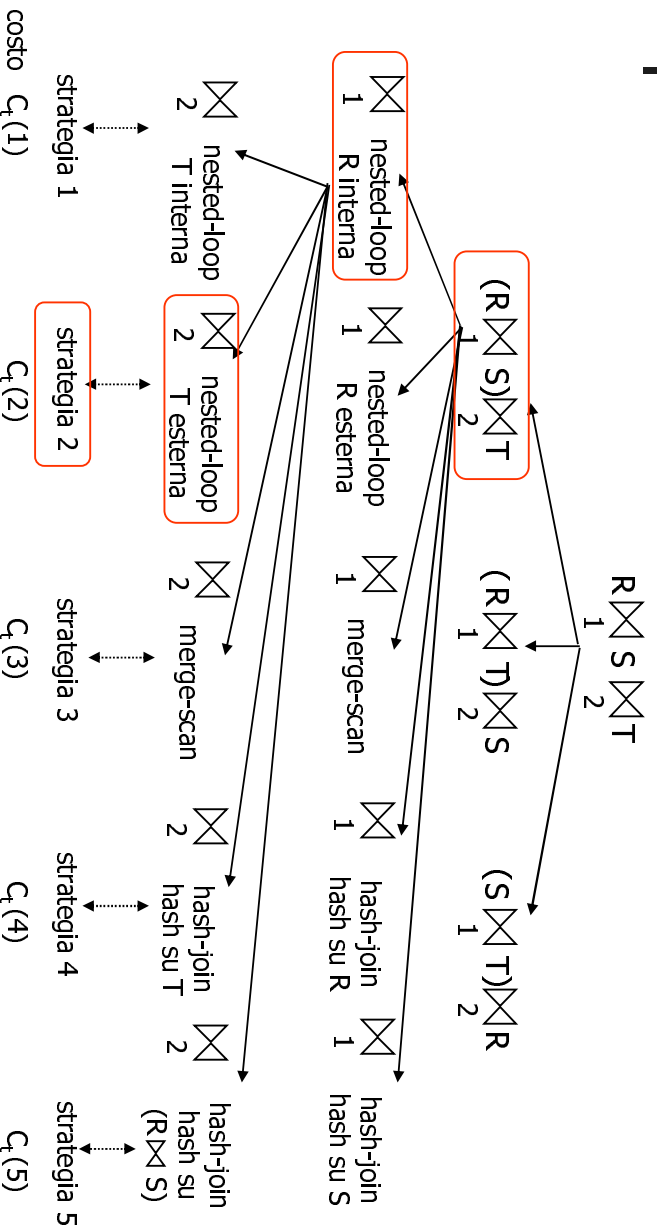
Basati su hashing

- i dati vengono **ricorsivamente** divisi in $M-1$ bucket R_1, \dots, R_n ($n > M$) con una funzione di hash fino quando la memoria non li contiene
- la funzione voluta viene calcolato separatamente su tutti i bucket R_1, \dots, R_n o coppie di bucket

Ottimizzazione basata sui costi di esecuzione

- Le dimensioni per l'ottimizzazione sono molte
 - tipologia dell'operazione di accesso ai dati (es. scan/indice)
 - ordine delle operazioni (es. ordine dei join)
 - modalità di realizzazione di una operazione (es. modalità di join)
 - livello in cui eseguire gli ordinamenti
- Si costruisce un albero delle alternative
 - ogni nodo corrisponde ad una scelta di un'opzione
 - ogni foglia rappresenta una strategia di esecuzione (descritta dal cammino dalla radice alla foglia)
 - Si dovrebbe scegliere il percorso radice-foglia con il costo minore

Esempio



Profili delle relazioni

- Informazioni quantitative relative alle caratteristiche delle tabelle
 - cardinalità della tabella T - $CARD(T)$
 - dimensione in byte delle tuple di T - $SIZE(T)$
 - dimensione in byte di ciascun attributo - $SIZE(A_j, T)$
 - numero di valori distinti di ciascun attributo - $VAL(A_j, T)$
 - i valori minimo e massimo di ciascun attributo - $MIN(A_j, T)$ $MAX(A_j, T)$
- I profili sono costruiti periodicamente per le tabelle del database
- I profili permettono di fare delle stime sulle dimensioni dei risultati intermedi delle interrogazioni
- Poiché il risultato di interrogazione è essa stessa una tabella temporanea, stimarne le dimensioni vuol dire calcolarne il profilo

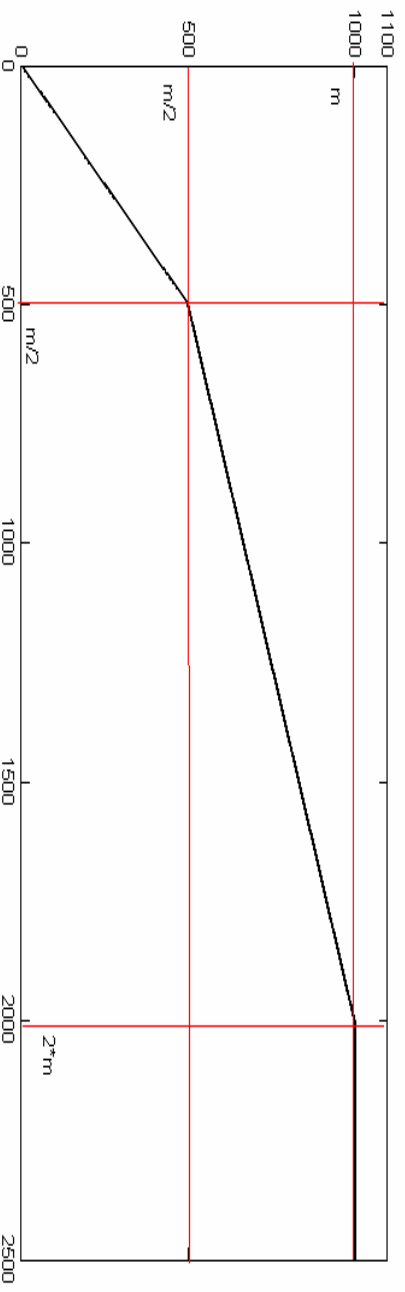
Profili per le selezioni: un'implementazione semplice

- Se $T' = \sigma_{A_i=v}(T)$
 - $CARD(T') = CARD(T)/VAL(A_i, T)$ (migliore stima in media)
 - $SIZE(T') = SIZE(T)$
 - $VAL(A_i, T')=1$
 - $VAL(A_j, T') = col(CARD(T), VAL(A_j, T), CARD(T'))$ $j \neq i$
 - numero di colori distinti selezionati scegliendo a caso $CARD(T')$ palline colorate da un insieme di $CARD(T)$ palline che hanno $VAL(A_j, T)$ colori diversi
 - $MAX(A_i, T') = MIN(A_i, T) = v$
 - $MAX(A_j, T') MIN(A_j, T') j \neq i$ mantengono i valori precedenti (ipotesi)
- $\sigma_{A_1 \wedge A_2}(T)$ è equivalente a $\sigma_{A_1} \sigma_{A_2}(T)$
- $\sigma_{A_1 \vee A_2}(T)$ è più difficile da stimare $CARD(T')$ - la somma dei risultati è una sovrastima

Esempio di implementazione di col

La seguente è un'implementazione possibile di $\text{col}(n,m,k)$

- Può essere implementata come
 - $\text{col}(n,m,k)=k$, se $k \leq m/2$
 - $\text{col}(n,m,k)=m$, $k > 2*m$
 - $\text{col}(n,m,k) = (k+m)/3$ se $m/2 < k < 2*m$



Profili per le proiezioni: un'implementazione

- Se $T' = \pi_L(T)$ $L = \{A_1, A_2, \dots, A_n\}$
 - $\text{CARD}(T') = \min(\text{CARD}(T), \prod_{i=1,n} \text{VAL}(A_i, T))$
 - $\text{SIZE}(T') = \sum_{i=1,n} \text{SIZE}(A_i, T)$
 - $\text{VAL}(A_i, T') \text{ MAX}(A_i, T')$ $\text{MAX}(A_i, T')$ mantengono i valori precedenti

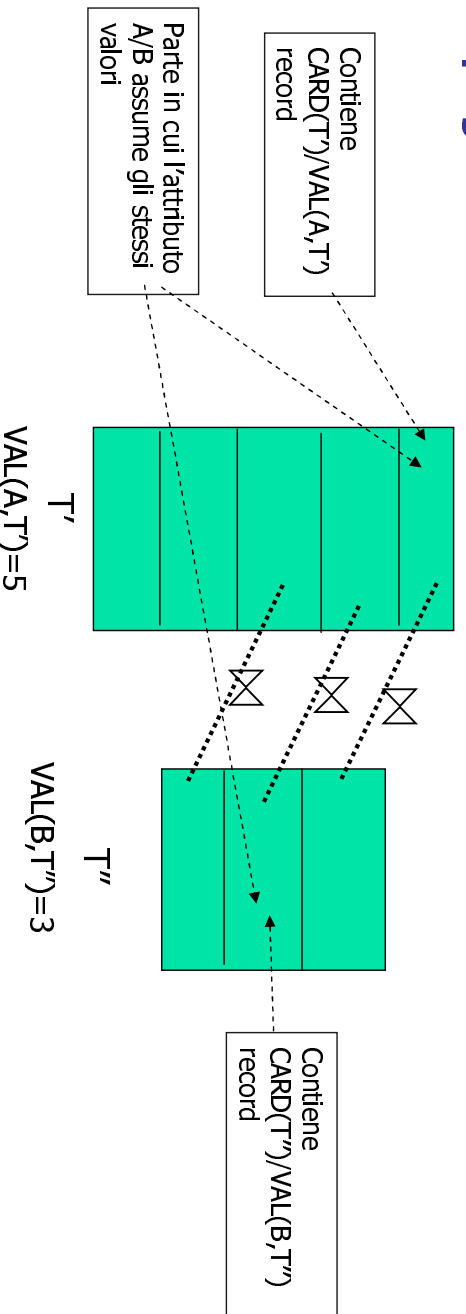
Profili per il join: un'implementazione

- $T^J = T' \bowtie_{A=B} T''$
 - $CARD(T^J) = CARD(T') * CARD(T'') / \max(VAL(A, T'), VAL(B, T''))$
 - $SIZE(T^J) = SIZE(T') + SIZE(T'')$
 - $VAL(A_i, T^J) \max(VAL(A_i, T'), VAL(A_i, T''))$ mantengono i valori precedenti nelle rispettive tabelle
- Tutte le formule assumono una distribuzione uniforme dei valori e l'assenza di correlazione fra le varie condizioni presenti in una interrogazione
- Questi dati approssimati sono comunque sufficienti a ottenere un'ottimizzazione ragionevole

Profili per il join

- $T^J = T' \bowtie_{A=B} T''$
 - $CARD(T^J) = CARD(T') * CARD(T'') / \max(VAL(A, T'), VAL(B, T''))$
 $= \min(VAL(A, T'), VAL(B, T'')) * (CARD(T') / VAL(A, T')) * (CARD(T'') / VAL(B, T''))$

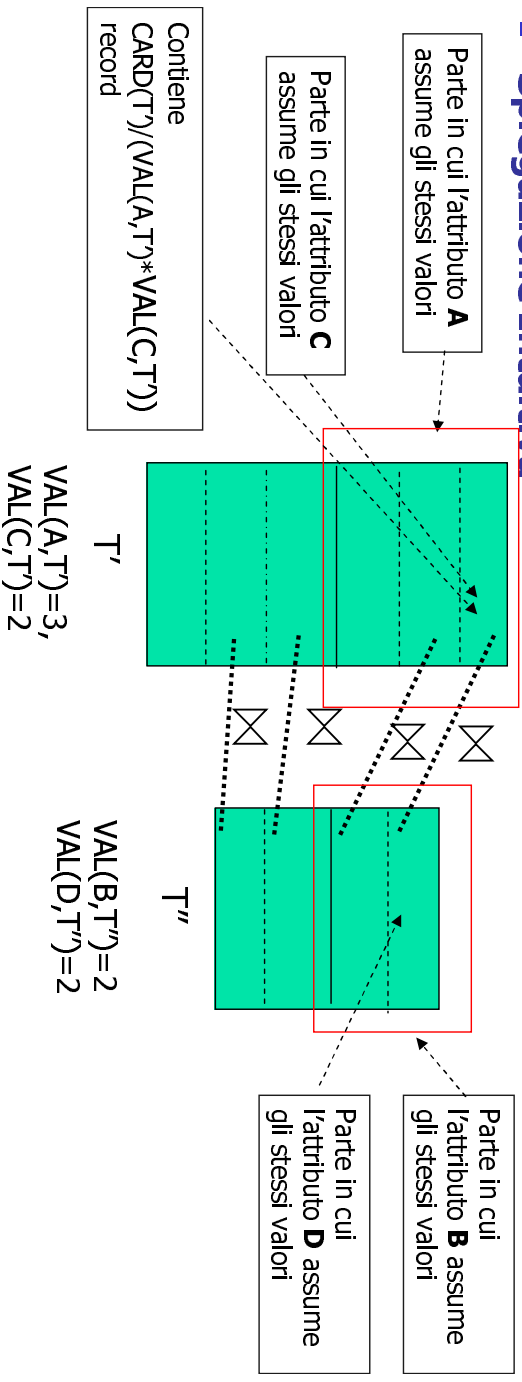
Spiegazione Intuitiva



Profili per il join: con più attributi

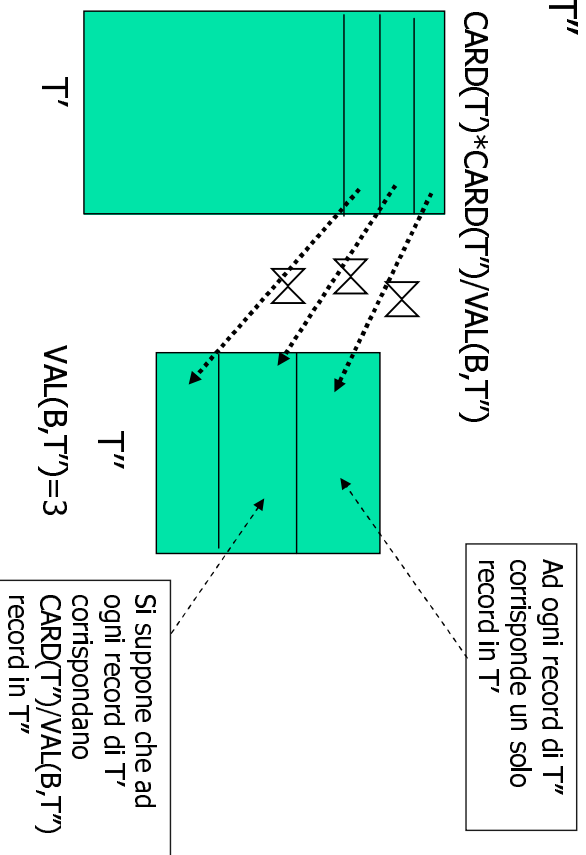
- $T^J = T' \bowtie_{A=B, C=D} T''$
 - $CARD(T^J) = CARD(T') * CARD(T'') / \max(VAL(A, T'), VAL(C, T'')) * \max(VAL(B, T'), VAL(D, T''))$

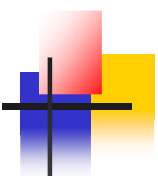
Spiegazione Intuitiva



Profili per il join: con tabelle slave e master

- Nel caso in A sia la chiave una tabella master T' e B la chiave esterna di una tabella slave T''
- $T^J = T' \bowtie_{A=B, T''} T''$
 - $CARD(T^J) = CARD(T') * CARD(T'') / VAL(B, T'')$





Progettazione fisica

- Definizione dei parametri fisici nel DBMS
- Scelta degli indici su ogni relazione
 - permettono di rendere più efficienti selezione e join
 - occorre scegliere gli attributi su cui crearli
 - creare un indice può essere costoso in spazio e tempo in modifica/inserimento
 - spesso le chiavi sono coinvolte in selezioni e join per cui conviene creare indici sui campi chiave (in alcuni DBMS sono creati automaticamente)
 - un indice può velocizzare anche le operazioni di ordinamento
 - su alcuni DBMS si può ottenere il piano degli accessi per verificare se gli indici sono utilizzati

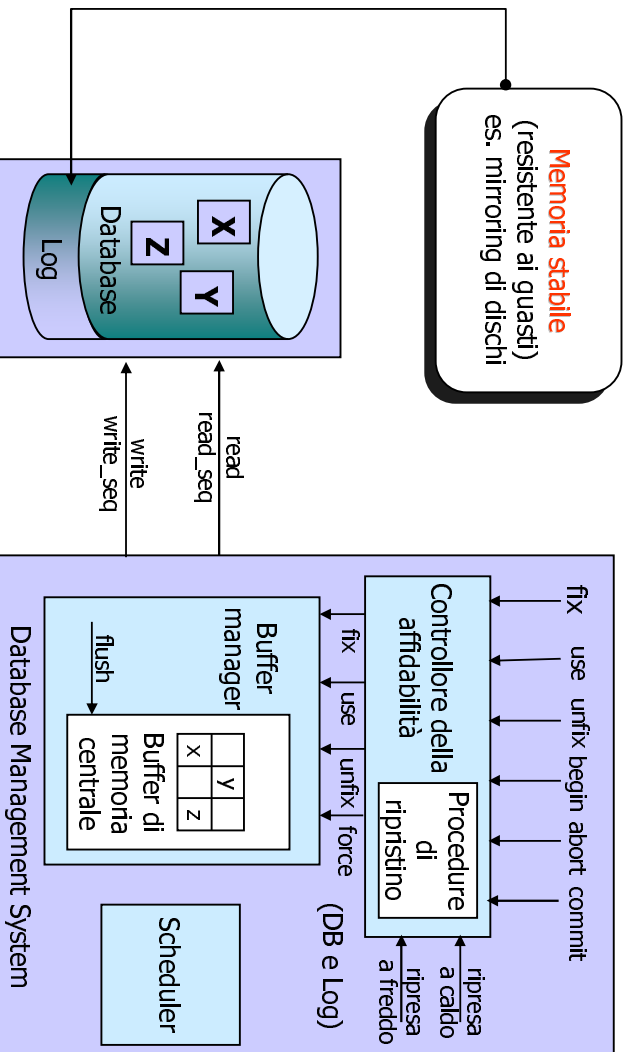


Controllo di affidabilità

Controllo di affidabilità

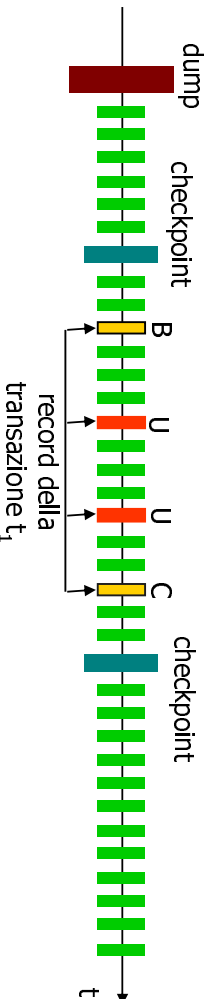
- Garantisce l'**atomicità** e la **persistenza** delle transazioni
- Si basa su un file di **log**
 - Il log registra le azioni (scritture) svolte dal DBMS
 - Il log permette di fare l'**undo** o il **redo** delle azioni
- Realizza i comandi transazionali
 - begin transaction (B)
 - commit work (C)
 - rollback work (A)
- Permette il ripristino in caso di malfunzionamenti (ripresa a caldo - ripresa a freddo)

Controllore di affidabilità



Organizzazione del file di log

- Il file di log è un **file sequenziale** su cui vengono registrate le azioni svolte dalle transazioni in ordine temporale



- **record di transazione**
 - record di begin (**B**)
 - record relativi alle operazioni effettuate (**I**nsert - **U**ppdate - **D**elete)
 - record di commit (**C**) o abort (**A**)
- **record di sistema** (Dump - Checkpoint)

Record di transazione

- **Begin** B(T) - **Commit** C(T) - **Abort** A(T)
Specificano l'identificativo T della transazione
- **Update** U(T,O,BS,AS)
Specificano la transazione T, l'oggetto O su cui si è fatto l'aggiornamento, il valore BS dell'oggetto prima della modifica (Before State) e quello AS dopo la modifica (After State)
- **Insert** I(T,O,AS)
Specificano la transazione T, l'oggetto O di cui si è fatto l'inserimento e il valore AS dopo l'inserimento
- **Delete** D(T,O,BS)
Specificano la transazione T, l'oggetto O di cui si è effettuata la cancellazione e il valore che aveva prima della cancellazione



Checkpoint e dump

- **Checkpoint** - $C(T_1, T_2, \dots, T_k)$
 - Operazione periodica per registrare tutte le transazioni attive
 - Si scrivono su disco le pagine relative a transazioni terminate con commit o abort (flush)
 - Non sono accettate operazioni di commit fino al termine del checkpoint
 - Gli effetti delle transazioni che hanno eseguito un commit o un abort sono rese persistenti
 - Si scrive nel log il record di checkpoint che contiene gli identificatori delle transazioni attive T_1, T_2, \dots, T_k
- **Dump** (backup) - DUMP
 - Copia completa della base di dati compiuta offline su supporto affidabile



Undo e redo

- I record di transazione permettono di disfare (undo) o rifare (redo) le rispettive azioni
- **Undo**
 - si ricopia in O il valore precedente BS
 - per annullare l'insert si cancella l'oggetto O
- **Redo**
 - si ricopia in O il valore AS
 - per ripetere la cancellazione basta cancellare O
- Vale l'**idempotenza**: ripetendo più volte lo stesso undo (redo) si ottiene lo stesso effetto di una sola ripetizione

Gestione delle transazioni: regole valide per ogni protocollo

■ Regola WAL (Write Ahead Log)

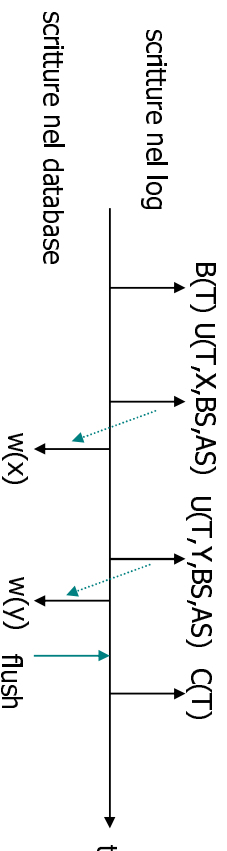
- La parte BS deve essere scritta nel log prima di effettuare la corrispondente operazione nella base di dati
- Viene mantenuto in modo affidabile il valore precedente alla scrittura
- Permette di fare l'undo delle scritture di una transazione che non ha fatto il commit
- In pratica il record di log è scritto completamente (BS e AS)

■ Regola di Commit-Precedenza

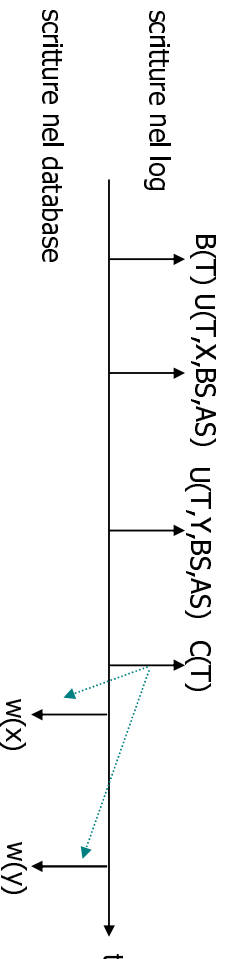
- La parte AS deve essere scritta nel log prima del commit
- Permette di fare il redo delle transazioni che sono completate col commit - la scrittura del record di commit rende effettiva la transazione
- In pratica il record di log è scritto completamente (BS e AS)

Protocolli di scrittura del log semplificata

- flush (force) prima del commit - non richiede il redo

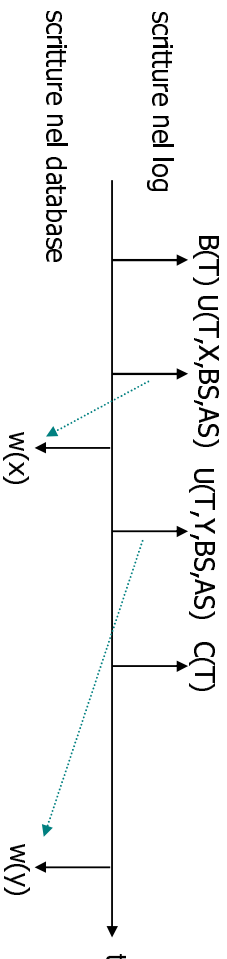


- flush (force) dopo il commit - non richiede undo



Protocolli di scrittura del log

- nessun vincolo sul momento della scrittura rispetto al commit (solo le regole wall e commit-precedenza)



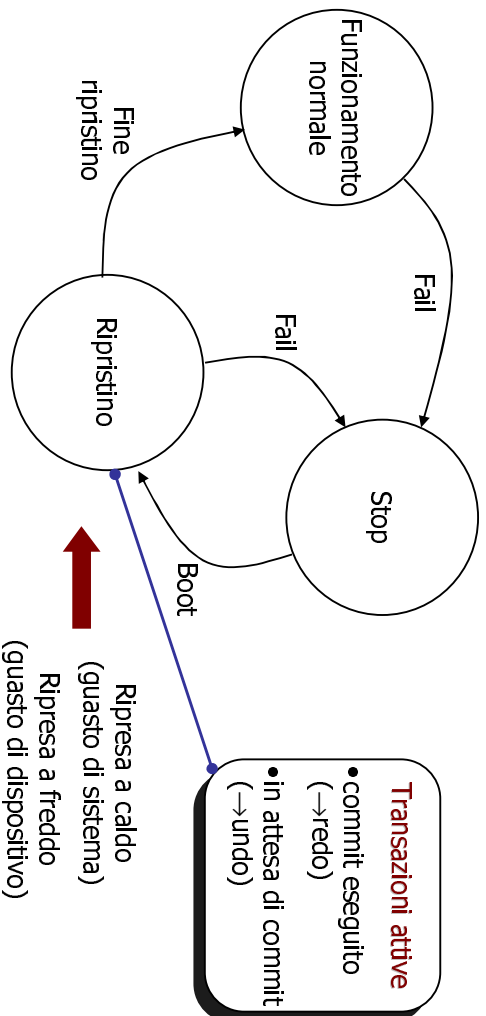
- Le operazioni di flush sono ottimizzate dal buffer manager
- Richiede redo e undo
- L'uso del file di log rappresenta un **carico aggiuntivo** per il sistema
- Devono essere adottate soluzioni efficienti per la scrittura del log

Tipologie di guasti

- **Guasti di sistema**
 - Bug del software, cali di tensione
 - Perdita del contenuto della memoria centrale (buffer)
 - Persistenza dei dati in memoria secondaria (dati e log)
- **Guasti di dispositivo**
 - Rottura dei dispositivi di memoria di massa
 - Si assume che il log sia salvato su memoria stabile (affidabile)
 - Si perde il contenuto della base di dati ma non del log
 - La perdita del log è un **evento catastrofico** senza recupero!

Modello fail-stop

- **Fail-stop**
quando si rileva un guasto il sistema arresta tutte le transazioni



Warm restart

- Si accede all'ultimo blocco del log e si ripercorre il log indietro fino al record di checkpoint
- Si determinano le transazioni da rifare (**REDO**) e da disfare (**UNDO**)
 - Inizializzazione
 - $UNDO = \{T_{transazioni\ attive\ al\ checkpoint}\}$
 - $REDO = \emptyset$
 - Scan del log in avanti
 - $B(T) \Rightarrow UNDO = UNDO \cup \{T\}$
 - $C(T) \Rightarrow REDO = REDO \cup \{T\}$ e $UNDO = UNDO / \{T\}$
 - Un'azione è nell'insieme $UNDO$ anche se la transazione è terminata con un record di abort $A(T)$

Warm restart

- Si ripercorre all'indietro il log fino all'azione più vecchia delle transazioni in UNDO e REDO
 - Si può anche andare oltre il checkpoint (se le transazioni attive al checkpoint hanno azioni precedenti)
 - Si dismano le azioni delle transazioni nell'insieme UNDO
 - Si applicano le azioni delle transazioni nell'insieme di REDO nell'ordine in cui sono nel log
- **Atomicità**: viene garantito che le transazioni in corso al momento del guasto non siano lasciate in uno stadio intermedio
- **Persistenza**: le transazioni che hanno fatto il commit sono completate rendendo permanenti le modifiche al database

Un esempio

- File di log che coinvolge le transazioni T_1, T_2, T_3, T_4, T_5
 $B(T_1) B(T_2) U(T_2, O_1, B_1, A_1) I(T_1, O_2, A_2) B(T_3) C(T_1) B(T_4)$
 $U(T_3, O_2, B_3, A_3) U(T_4, O_3, B_4, A_4) CK(T_2, T_3, T_4) C(T_4) B(T_5) U(T_3, O_3, B_5, A_5)$
 $U(T_5, O_4, B_6, A_6) D(T_3, O_5, B_7) C(T_5) I(T_2, O_6, A_8) FAIL$
- Si accede al checkpoint precedente al FAIL
 - UNDO = $\{T_2, T_3, T_4\}$ e REDO = $\{\}$
- Si aggiornano gli insiemi REDO e UNDO
 - $C(T_4)$: UNDO = $\{T_2, T_3\}$ e REDO = $\{T_4\}$
 - $B(T_5)$: UNDO = $\{T_2, T_3, T_5\}$ e REDO = $\{T_4\}$
 - $C(T_5)$: UNDO = $\{T_2, T_3\}$ e REDO = $\{T_4, T_5\}$

Un esempio

$B(T_1)$ $B(T_2)$ $U(T_2, O_1, B_1, A_1)$ $I(T_1, O_2, A_2)$ $B(T_3)$ $C(T_1)$ $B(T_4)$
 $U(T_3, O_2, B_3, A_3)$ $U(T_4, O_3, B_4, A_4)$ $CK(T_2, T_3, T_4)$ $C(T_4)$ $B(T_5)$ $U(T_3, O_3, B_5, A_5)$
 $U(T_5, O_4, B_6, A_6)$ $D(T_3, O_5, B_7)$ $C(T_5)$ $I(T_2, O_6, A_8)$ **FAIL**

- UNDO
 - T_2 : Delete(O_6) T_3 : Insert($O_5=B_7$); $O_3=B_5$; $O_2 = B_3$ T_2 : $O_1 = B_1$
- REDO
 - T_4 : $O_3 = A_4$ T_5 : $O_4 = A_6$

Cold restart

- Il guasto consiste in un deterioramento di una parte del database
- I passi del cold restart sono
 - Si ripristina la parte deteriorata dall'ultimo **backup** (dump)
 - Si accede al record di dump più recente nel log
 - Si ripercorre in avanti il log ripetendo tutte le azioni per ripristinare la parte deteriorata
 - Si esegue una ripresa a caldo

Backup

I backup

- servono in caso di guasti con **perdita dei dati**
- riguardano
 - **i dati**: si copia il contenuto dell'archivio (dopo si puo' tagliare il log)
 - **il log**: permette di ricostruire lo stato della base di dati ad un qualsiasi istante
- sono di due tipi
 - **completi**:
 - si copia tutto il contenuto dell'archivio
 - **il backup e' piu' lento**
 - **incrementali**:
 - si copia solo i cambiamenti dall'ultimo backup
 - per ripristino occorre prima ripristinare un backup completo e poi i backup incrementali successivi
 - **il ripristino è piu' lento**
- Tipicamente si alternano backup completi a backup incrementali:
(es. completo ogni settimana, incrementale ogni giorno)

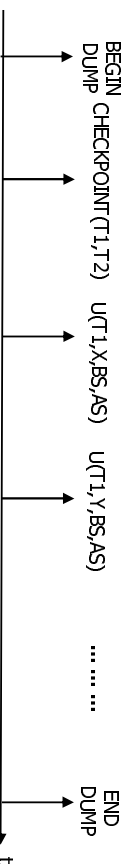
Backup on line

La soluzione piu' semplice

- disconnettere tutti gli utenti e fare il backup
- deve esistere un periodo sufficientemente lungo durante il quale il DBMS puo' essere fermato (ad es., la notte)

Backup con transazioni attive

- durante il backup
 - si scrive sul log il momento di inizio
 - si fa un checkpoint
 - si copiano i dati
 - si scrive sul log il momento di fine del backup
 - si copia il log



Backup on line II

- durante il ripristino
 - si ricopia dal backup i dati andati persi
 - si applica una ripartenza a caldo, come se ci fosse stato un fallimento all'istante della fine del backup
 - l'archivio si ritrova nello stato del momento in cui è finito il backup (per metterlo nello stato del momento in cui è iniziato il backup occorre non usare i commit fra l'inizio e la fine del backup)

B(T1) B(T2) U(T2,O1,B1,A1) I(T1,O2,A2) C(T1) B(T3) U(T3,O3,B4,A4)
BEGIN_DUMP CK(T2,T3) C(T3) B(T4) U(T4,O4,B6,A6) C(T4) I(T2,O6,A8) END_DUMP

Log

UNDO = {T2} e REDO = {T3, T4} ←

Ripristino

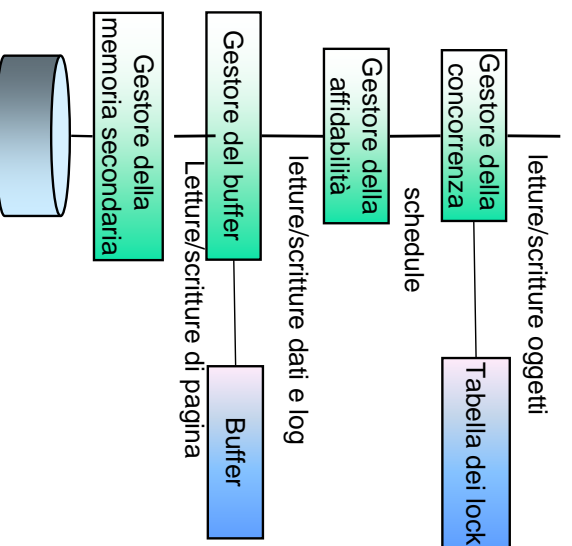
Gestione della concorrenza

Controllo di concorrenza

- Accesso al DBMS da parte di più utenti
- Il carico si misura in **tps** (transactions per second)
 - 10-10² tps in sistemi bancari
 - 10³ tps sistemi di prenotazione dei voli, gestori delle carte di credito
- La concorrenza permette un uso efficiente del DBMS massimizzando il numero di transazioni eseguite al secondo e minimizzando i tempi di risposta
- Si considerano le **operazioni di ingresso/uscita di basso livello**
 - trasferimenti fra la memoria di massa e la memoria centrale di blocchi di dati (pagine) - letture/scritture
- Il controllore della concorrenza è uno **scheduler** determina se le richieste possono essere soddisfatte

Architettura

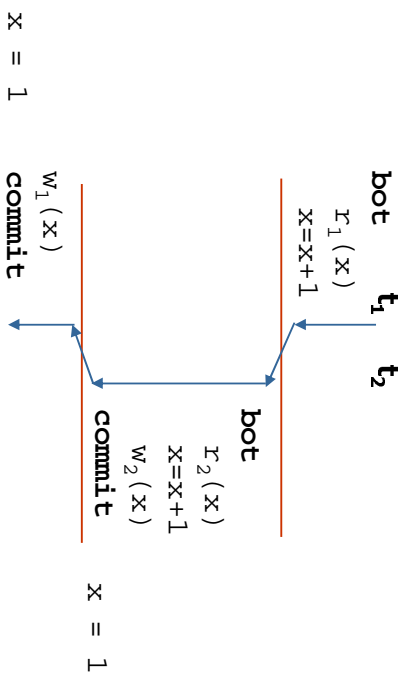
- gestore delle concorrenza (scheduler)
 - garantisce che le transazioni concorrenti non interferiscano. Può usare una tabella dei lock
- gestore dell'affidabilità
 - gestisce le strategie di log e ripristino
- gestore del buffer
 - gestisce i trasferimenti da disco a memoria primaria



Nota che tali moduli non corrispondono necessariamente a moduli di un DBMS reale

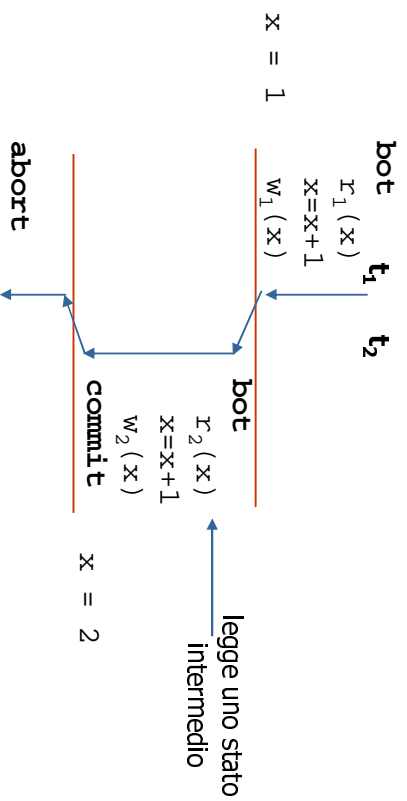
Perdita di aggiornamento

- Si considerano due transazioni che operano sullo stesso dato
 - $t_1 : r(x), x=x+1, w(x)$
 - $t_2 : r(x), x=x+1, w(x)$
- se inizialmente $x=0$, dopo l'esecuzione seriale $x=2$



Lecture sporche

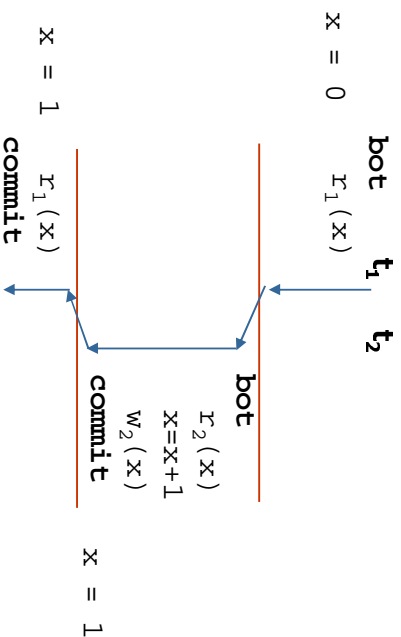
- Si considerano le due transazioni t_1 e t_2 con fallimento di t_1



- Per il fallimento di t_1 , dovrebbe essere $x=1$ alla fine
- L'abort di t_1 dovrebbe causare il fallimento di t_2 (effetto domino)

Lecture inconsistenti

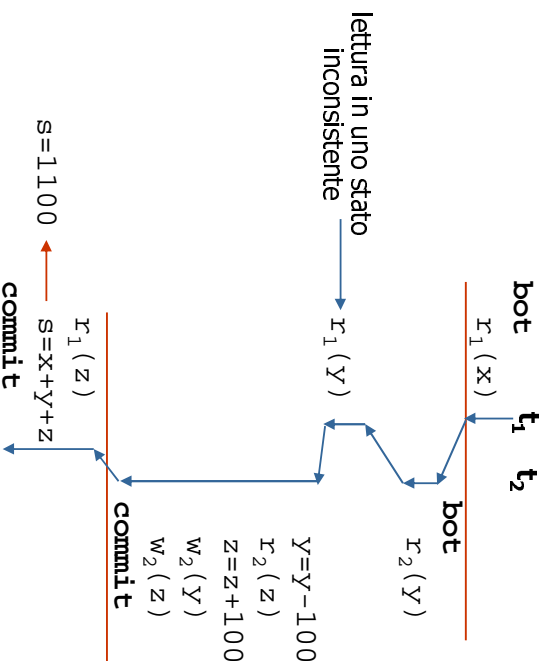
- La transazione t_1 ripete la lettura di x in istanti successivi



- All'interno della stessa transazione il valore letto non deve risentire dell'effetto di altre transazioni

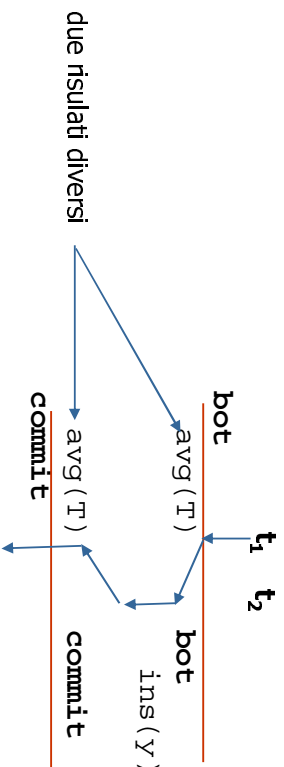
Aggiornamento fantasma

- Si suppone che esista il vincolo di integrità $x+y+z=1000$



Inserimento fantasma

- Una transizione calcola due volte una valore aggregato $avg(T)$ mentre l'altra inserisce un nuovo dato $ins(y)$



- Esiste anche la rimozione fantasma

Teoria della concorrenza

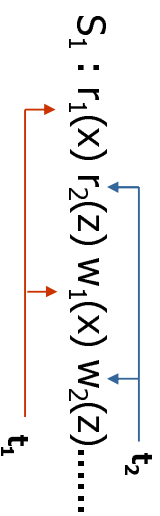
- Transazione** = sequenza di azioni di scrittura o lettura
- Ogni transazione ha un unico identificatore assegnato
- Ogni transazione è compresa fra i comandi **begin transaction** e **end transaction** (omessi)
- Un esempio è



- Il controllo di concorrenza accetta/rifiuta esecuzioni concorrenti durante l'evoluzione delle transizioni (senza saperne l'esito)

Schedule

- Uno schedule è una sequenza di operazioni di lettura/scrittura corrispondente a transazioni concorrenti



- In teoria, si considerano solo transazioni che terminano con un commit (**commit-proiezioni**)
 - In pratica, il controllore della concorrenza deve occuparsi anche delle transizioni che terminano con un abort
- Obiettivo del controllo di concorrenza è di riuscire a generare solo schedule che non causano anomalie

Schedule seriali e serializzabili

- Uno **schedule seriale** è uno schedule in cui tutte le operazioni di ogni transazione compaiono in sequenza
 - $S_2 : \underbrace{r_0(X) \ r_0(Y) \ w_0(X)}_{t_0} \underbrace{r_1(Y) \ r_1(X) \ w_1(Y) \ r_2(X) \ r_2(Y) \ r_2(Z) \ w_2(Z)}_{t_2}$
 - le transazioni godono della proprietà dell'isolamento
- Uno schedule è **serializzabile** se produce lo stesso risultato di uno schedule seriale delle stesse transazioni
 - l'utente non può distinguere fra schedule seriali e schedule serializzabili
- gli schedule serializzabili
 - si comportano come schedule seriali
 - sono più efficienti degli schedule seriali

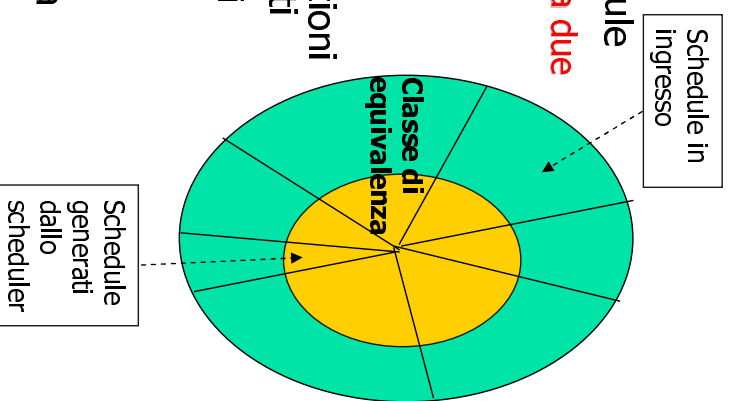
Schedule seriali e serializzabili II

La teoria

- si introduce la nozione di equivalenza di schedule
 - **view-equivalenza**, **conflict-equivalenza**, **locking a due fasi**, controllo basato su **timestamp**
- ciascuna equivalenza identifica una classe di schedule accettabili

La pratica

- uno scheduler trasforma la sequenza di transizioni in ingresso in schedule serializzabili equivalenti
 - fa in modo che le transizione serializzabili non si verifichino oppure
 - uccide le transazioni non serializzabili
- Il progetto dello scheduler deve bilanciare l'efficienza delle transazioni e l'efficienza quella dello scheduler



View-equivalenza

Definizioni

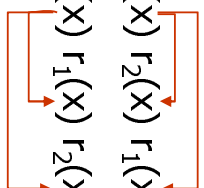
- $r_i(x)$ **legge-da** $w_j(x)$ ($\text{legge}(r_i(x), w_j(x))$)
se $w_j(x)$ precede $r_i(x)$ e non esiste $w_k(x)$ compreso fra $w_j(x)$ e $r_i(x)$
..... $w_j(x)$ $r_i(x)$

- $w_i(x)$ è una **scrittura finale** se è l'ultima scrittura di x nello schedule
- Due schedule sono **view-equivalenti** ($S_i \approx_v S_j$) se su di essi sono definite le stesse relazioni legge-da e le stesse scritture finali
- Uno schedule è **view-serIALIZZABILE** se è view-equivalente ad un generico schedule seriale
- L'insieme degli schedule view-serIALIZZABILI è detto **VSR**

Un esempio: schedule view-equivalenti

$S_3 : w_0(x) \ r_2(x) \ r_1(x) \ w_2(x) \ w_2(z)$

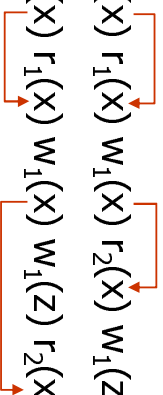
$S_4 : w_0(x) \ r_1(x) \ r_2(x) \ w_2(x) \ w_2(z)$



S_4 è uno schedule seriale; S_3 è serializzabile

$S_5 : w_0(x) \ r_1(x) \ w_1(x) \ r_2(x) \ w_1(z)$

$S_6 : w_0(x) \ r_1(x) \ w_1(x) \ w_1(z) \ r_2(x)$



S_6 è uno schedule seriale; S_5 è serializzabile

Schedule non serializzabili

- Perdita di aggiornamento

$S_7 : r_1(x) \ r_2(x) \ w_2(x) \ w_1(x)$
 $r_1(x) \ w_1(x) \ r_2(x) \ w_2(x)$
 $r_2(x) \ w_2(x) \ r_1(x) \ w_1(x)$

- Lettura inconsistente

$S_8 : r_1(x) \ r_2(x) \ w_2(x) \ r_1(x)$
 $r_1(x) \ r_1(x) \ r_2(x) \ w_2(x)$
 $r_2(x) \ w_2(x) \ r_1(x) \ r_1(x)$

Schedule seriali disponibili

- Aggiornamento fantasma

$S_9 : r_1(x) \ r_1(y) \ r_2(z) \ r_2(y) \ w_2(y) \ w_2(z) \ r_1(z)$



Complessità

- Decidere se due schedule sono view-equivalenti ha complessità **lineare**
 - E' efficiente confrontare fra loro due schedule
- Decidere se un generico schedule è view-serializzabile è un problema **NP-completo**
 - Occorre confrontare lo schedule con tutti gli schedule seriali ottenuti permutando le transazioni
 - Non si può usare questa nozione per verificare la serializzabilità



Conflitti

- L'azione a_i è in **conflitto** con l'azione a_j ($i \neq j$) se operano sullo stesso oggetto ed almeno una è un write
 - Conflitti lettura-scrittura (rw o wr)
 - Conflitti scrittura-scrittura (ww)
- Intuitivamente:
 - due azioni sono in conflitto se non si può cambiare l'ordine senza effetti (Si considerano solo le due azioni)

Esempi: si può scambiare l'ordine delle azioni senza effetti

- $r_i(x) \dots r_j(y)$ non è mai un conflitto anche se $x=y$
Le letture non modificano i valori
- $r_i(x) \dots w_j(y) \dots r_i(x)$ o $w_j(y) \dots w_i(x)$ non è un conflitto se $x \neq y$
La modifica su y non cambia la lettura/scrittura su x



Conflitti

Esempi: **non si può scambiare l'ordine delle seguenti azioni**

- $w_i(x) \dots w_j(x)$ è un conflitto
Il valore di x dopo l'esecuzione è definito da t_j . Scambiando l'ordine sarebbe invece definito da t_i
- $w_i(x) \dots r_j(x)$ è un conflitto
Il valore di x che deve essere letto è quello scritto da t_j
- $r_i(x) \dots w_j(x)$ è un conflitto
Il valore di x che deve essere letto è quello precedente alla scrittura da parte di t_j

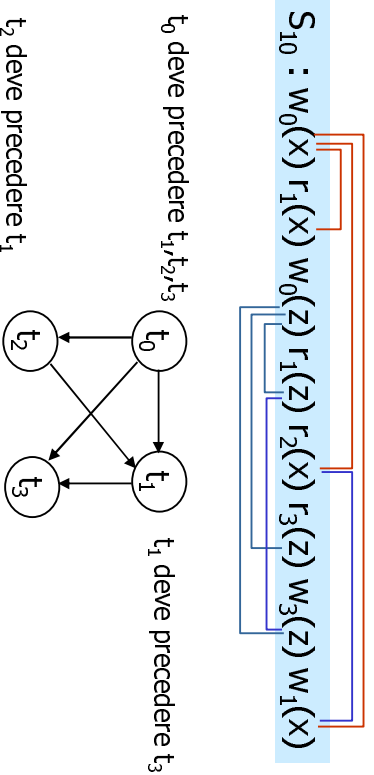


Conflict-equivalenza

- Si possono fare scambi che non cambiano i conflitti presenti
- Due schedule sono **conflict-equivalenti** ($S_i \approx_c S_j$) se
 - i due schedule presentano le stesse operazioni
 - ogni coppia di operazioni in conflitto è nello stesso ordine in entrambi gli schedule
- Uno schedule è **conflict-serIALIZZABILE** se è conflict-equivalente ad un generico schedule seriale
- Se due azioni sono in conflitto le corrispondenti transazioni nello schedule serializzato devono essere nello stesso ordine
- L'insieme degli schedule conflict-serIALIZZABILI è detto CSR
- Risulta $CSR \subset VSR$

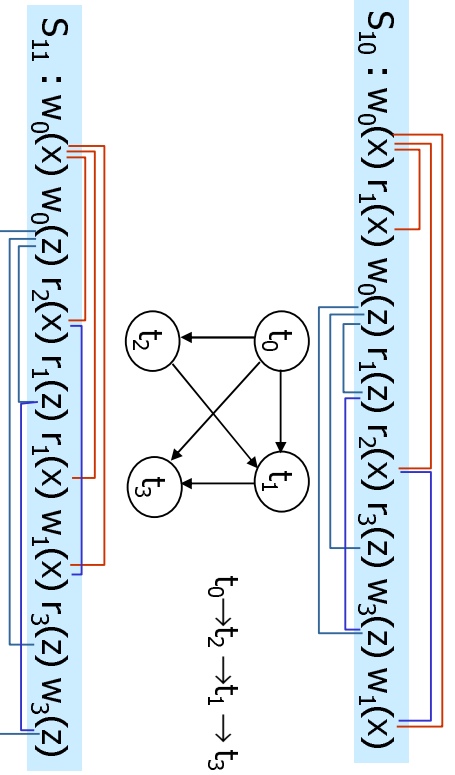
Verifica della conflict-serializzabilità

- Si costruisce il grafo dei conflitti (o di precedenza)
 - ogni nodo corrisponde ad una transazione
 - Un arco fra t_i e t_j indica che c'è almeno un conflitto fra un'azione a_i e un'azione a_j tali che a_i precede a_j



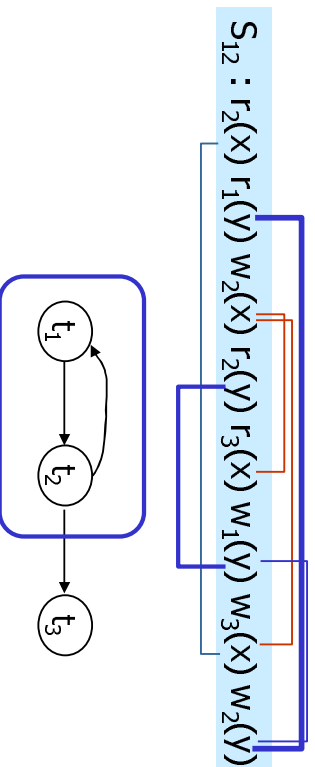
Conflict-serializzabilità

- Uno schedule è CSR se e solo se il grafo è aciclico
- La serializzazione è data da un ordinamento topologico dei nodi
- La complessità è lineare nel numero dei nodi del grafo (transazioni)



Conflict-serializzabilità

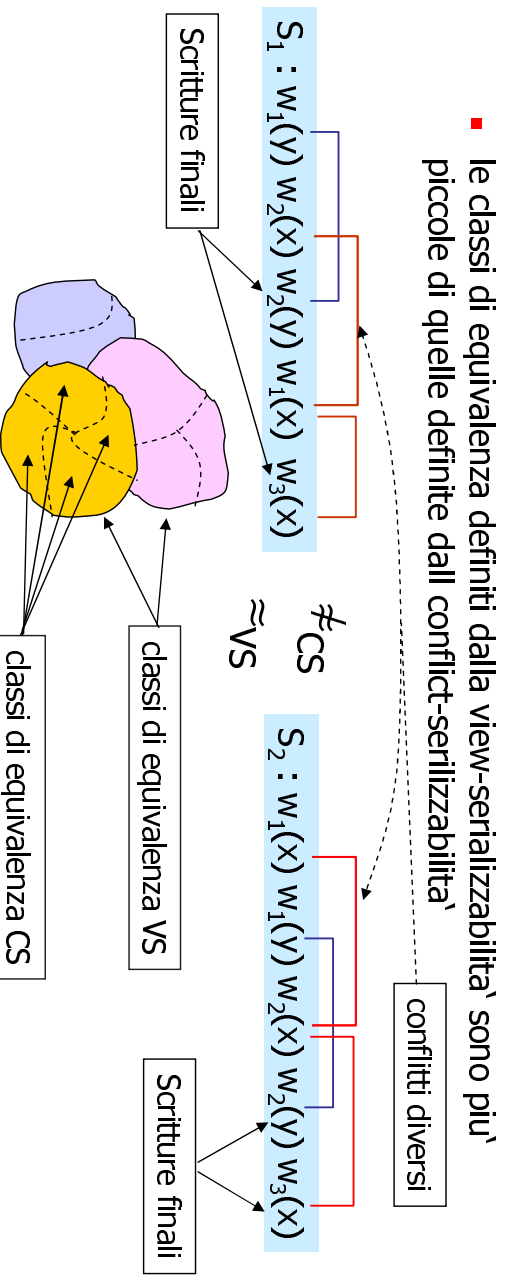
- Un esempio di schedule non conflict-serializzabile



- Non si può decidere se mettere prima t_1 o t_2
- Questo mostra perché lo schedule non è conflict-serializzabile se ci sono cicli nel grafo delle precedenze

Conflict e view serializzabilità

- Uno schedule conflict-serializzabile e' view-serializzabile ($CSR \subset VSR$)
- Non è vero il viceversa: ci sono schedule view-serializzabili che non sono conflict-serializzabili



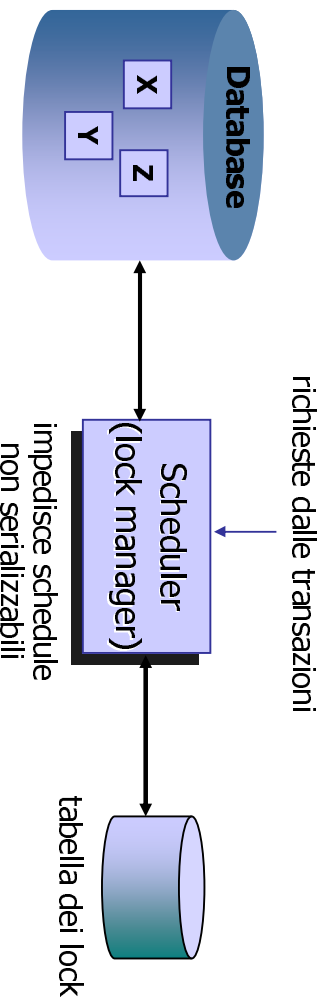
Implementazione dello scheduler

Esistono due approcci diversi all'implementazioni dello scheduler

- Approcci basati sul **controllo delle transazioni**
 - si controlla le transazioni evitando **il verificarsi delle anomalie**: lo scheduler genera solo schedule serializzabili
 - tipicamente questi approcci sono basati su locking
- Approcci **ottimistici**
 - assumono che **tutto andrà bene**, se poi si verificano anomalie, lo scheduler interviene
 - l'intervento consiste nell'uccidere una delle transazioni colpevoli dell'anomalia

Locks

- L'uso del locking è molto comune nei DBMS commerciali
- Le operazioni di lettura e scrittura sono protette dalle primitive
 - **r_lock** - read lock
 - **w_lock** - write lock
 - **unlock**



Tipi di lock

- Ogni lock è applicato ad uno specifico dato del database
- I vincoli da rispettare sono
 - Ogni lettura è preceduta da un **r_lock** e seguita da un **unlock**
Il lock è **condiviso** dato che sullo stesso dato possono essere attivi più lock di questo tipo (tipo S)
 - Ogni scrittura è preceduta da un **w_lock** e seguita da un **unlock**
Il lock è **esclusivo** perché non può coesistere con altri lock sullo stesso dato (tipo X)
- Una transazione che rispetta queste regole si dice **ben formata rispetto al locking**
- In genere le richieste di lock e unlock sono inserite in modo trasparente

Richieste di lock

- Quando la richiesta di lock è concessa, la transazione **acquisisce** la corrispondente risorsa
- Con l'esecuzione di unlock la risorsa è **rilasciata**
- Quando la richiesta di lock non viene concessa la transazione richiedente viene messa in **stato di attesa**

Richiesta	Stato della risorsa		
	libero	S	X
S	OK/S	OK/S	NO/X
X	OK/X	NO/X	NO/X
Unlock	Error	OK/dipende	OK/libero

Si: la risorsa viene concessa

No: la transazione è bloccata

Tabella di compatibilità

libero se non ci sono più richieste S

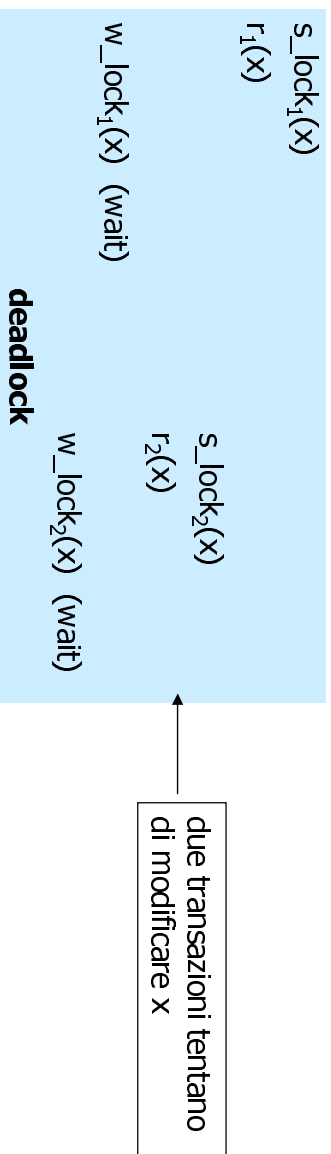
Aumentare il livello di lock

Se si vuol prima leggere e poi scrivere un oggetto

- si acquisisce prima un lock condiviso poi si incrementa a lock esclusivo

$s_lock(x)$ $r(x)$ $w_lock(x)$ $w(x)$ $unlock(x)$

- questa strategia aumenta la probabilita' di deadlock



Lock di tipo update

- $u_lock_t(x)$: lock di tipo update (tipo U)
 - fornisce il privilegio solo di lettura su x alla transazione t_i
 - può essere trasformato in un lock esclusivo (write)
 - una volta che è attivo su x non si possono aggiungere ulteriori lock attivi (le transazioni corrispondenti si mettono in wait)

Richiesta	Stato		
	S	X	U
S	OK/S	No	No
X	No	No	No
U	OK/U	No	No

Si: la risorsa viene concessa

No: la transazione è bloccata

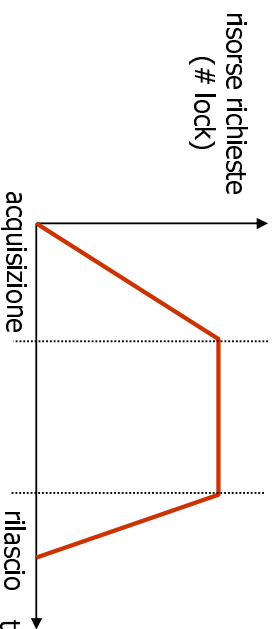
Tabella di compatibilità

Locking a due fasi

- Per garantire che le transazioni seguano uno schedule serializzabile si introduce il **two phase locking** (2PL)

Una transazione dopo aver rilasciato un lock non può acquisirne altri

- Perché 2 fasi
 - prima si acquisiscono i lock per le risorse necessarie (fase crescente)
 - poi si rilasciano i lock acquisiti (fase calante)



2PL e CSR

- Ogni schedule che rispetta il protocollo 2PL è anche serializzabile rispetto alla conflict-equivalenza **2PL \subset CSR**

- Dimostrazione per assurdo

Se $S \in 2PL$ e $S \notin CSR$, il grafo dei conflitti è ciclico

$$t_1 \rightarrow t_2 \rightarrow \dots \rightarrow t_n \rightarrow \dots \rightarrow t_1$$

- esiste una risorsa **R1** su cui t_1 e t_2 operano entrambe in modo conflittuale
- t_2 può proseguire solo quando t_1 rilascia il lock sulla risorsa
- iterando per ogni i.....
- esiste una risorsa **Rn** su cui t_n e t_1 operano entrambe in modo conflittuale
- Affinché t_1 possa procedere è necessario che acquisisca il lock sulla risorsa **Rn** rilasciata da t_n
- La transazione t_1 rilascia un risorsa (**R1**) prima di acquisirne un'altra (**Rn**), ovvero lo schedule non può essere 2PL

CSR e 2PL

- Uno schedule che è conflict-serializzabile non è detto che sia 2PL

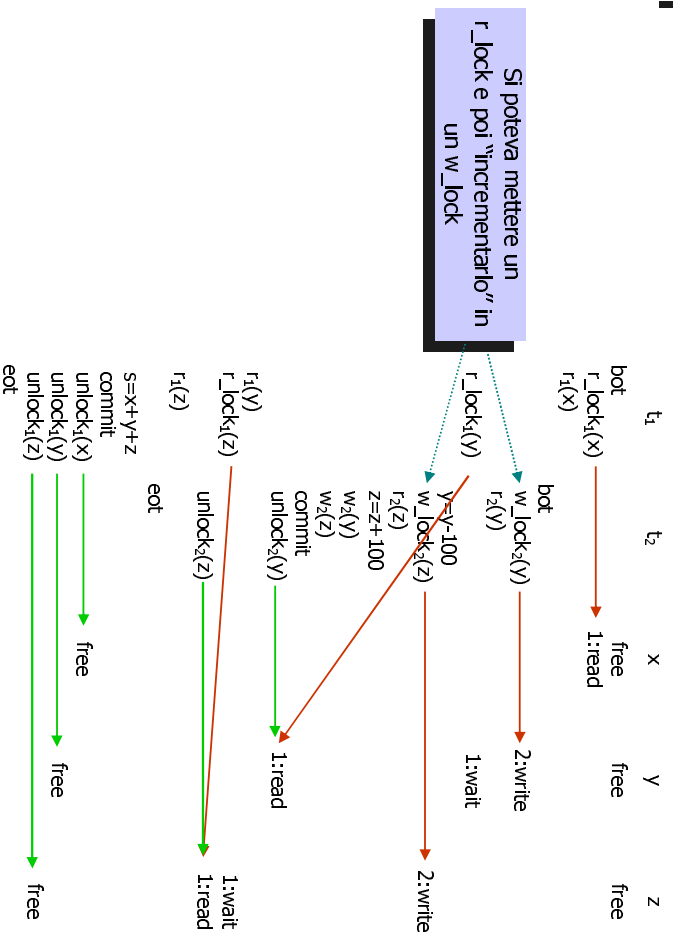
$S_{13} : r_1(x) \ w_1(x) \ r_2(x) \ w_2(x) \ r_3(y) \ w_1(y)$

$t_3 \rightarrow t_1 \rightarrow t_2$

$S_{14} : r_3(y) \ r_1(x) \ w_1(x) \ w_1(y) \ r_2(x) \ w_2(x)$

- Non è 2PL perché in S_{13} la transazione t_1 deve liberare il lock su x e poi richiedere il lock su y

Modifica fantasma: soluzione



Locking a 2 fasi "stretto"

- Si rimuove l'ipotesi che tutte le transazioni vadano a buon fine (commit-proiezione)

I lock di una transazione possono essere rilasciati solo dopo aver effettuato correttamente le operazioni di commit/abort

- I lock vengono rilasciati solo al termine della transazione dopo che ogni dato è in uno stato consistente
- E' utilizzato nei DBMS commerciali
- Elimina l'anomalia della lettura sporca (dovuta alla presenza degli abort)

Gestione dei lock

- Le operazioni di gestione dei lock hanno in genere il seguente prototipo
 - `r_lock(T,x,errcode,timeout)`
 - `w_lock(T,x,errcode,timeout)`
 - `unlock(T,x)`
- **T**: identificatore della transazione
- **x**: risorsa su cui si richiede/rilascia il lock
- **errcode**: codice che indica l'esito della richiesta (0 richiesta eseguita correttamente, diverso da 0 in caso di errore)
- **timeout**: tempo massimo che si attende per ottenere il lock sulla risorsa (se scade errcode assume il valore opportuno)
- Lo scheduler gestisce i lock mantenendo in memoria primaria delle strutture dati (**tabelle** **dei lock** e **code** di processi in attesa)

Gestione dei lock

- Se la richiesta può essere soddisfatta
 - il lock manager cambia lo stato della risorsa in tabelle dei lock
 - viene restituito il controllo al processo che ha effettuato la richiesta
- Se la richiesta non può essere soddisfatta
 - il processo richiedente viene inserito in una coda associata alla risorsa
 - il processo richiedente viene sospeso
 - quando la risorsa viene rilasciata si controlla se ci sono processi in attesa e nel caso si concede la risorsa al processo in testa alla coda
- Se scatta un timeout
 - la transizione fallisce e si esegue un rollback
 - si prova a richiedere nuovamente il lock

Tabelle dei lock

- Sono mantenute in memoria principale per motivi di efficienza
- Ad ogni oggetto è associato lo stato

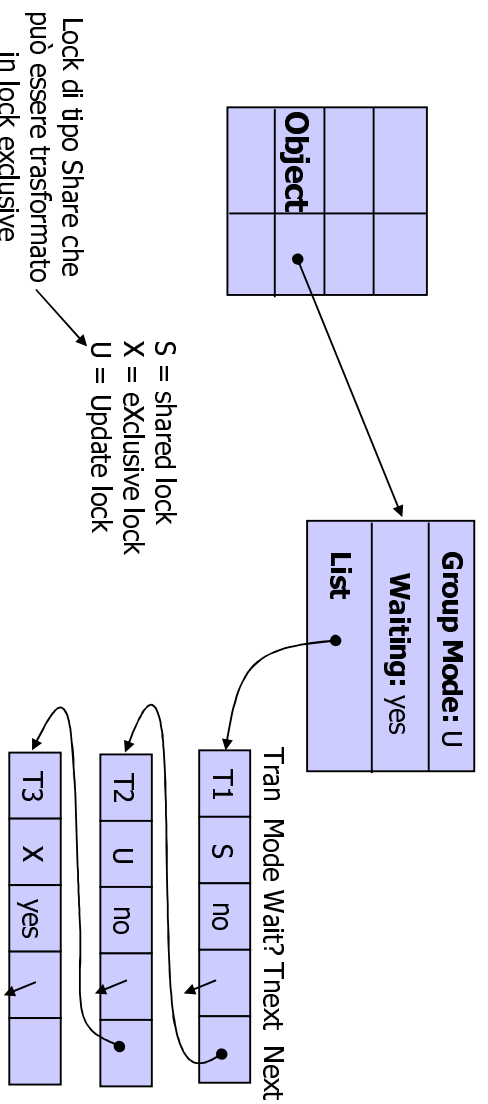
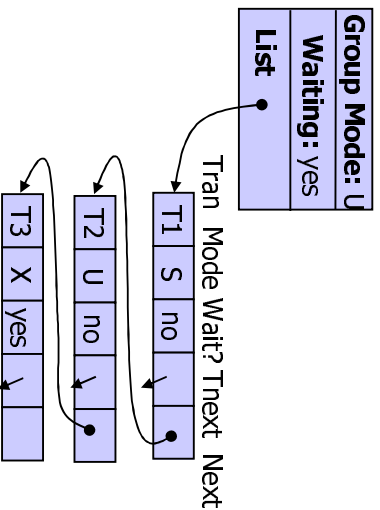
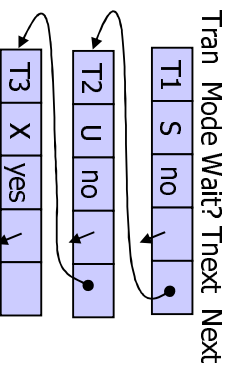


Tabella dei lock



- Il **group mode** individua la condizione più stringente attiva sulla risorsa
 - S se sono attivi solo shared lock
 - U se c'è un update lock e forse uno o più shared lock
 - X se c'è un exclusive lock
- Il bit **Waiting** indica se c'è almeno una transazione in attesa sulla risorsa
- List** contiene tutte le transazioni che al momento possiedono il lock o che sono in attesa di ottenerlo

Tabella dei lock



- Tran** è l'identificatore della transazione che detiene o attende un lock
- Mode** indica il tipo di lock (S X U)
- Wait?** è un flag che indica se la transazione detiene o attende un lock
- Tnext** permette di collegare i lock relativi alla stessa transazione. Questa lista può essere usata quando la transazione esegue un commit o un abort per rilasciare tutti i lock

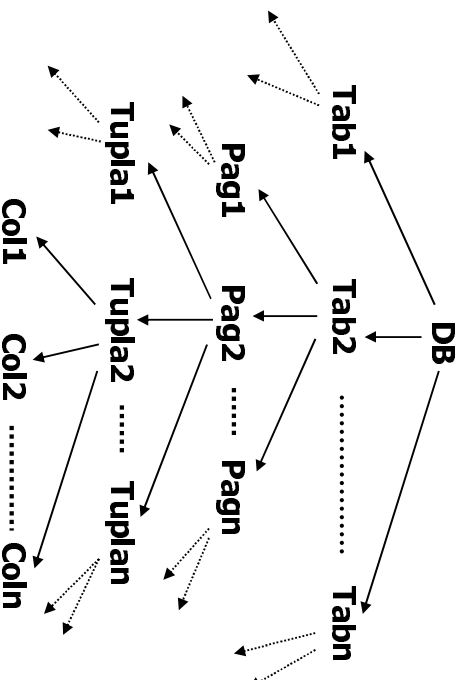
Granuralita' di locking

Quali sono gli oggetti su cui si deve porre un lock ?

- L'intera tabella
 - Per eliminare il problema dell'inserimento fantasma
 - Per un'interrogazione con operatore aggregato
- Alcuni blocchi
 - operazioni di modifica su un range (record consecutivi)
- Le tuple
 - la modifica di una tupla
- Il campo
 - La modifica di uno o pochi campi

Granuralita' di locking II

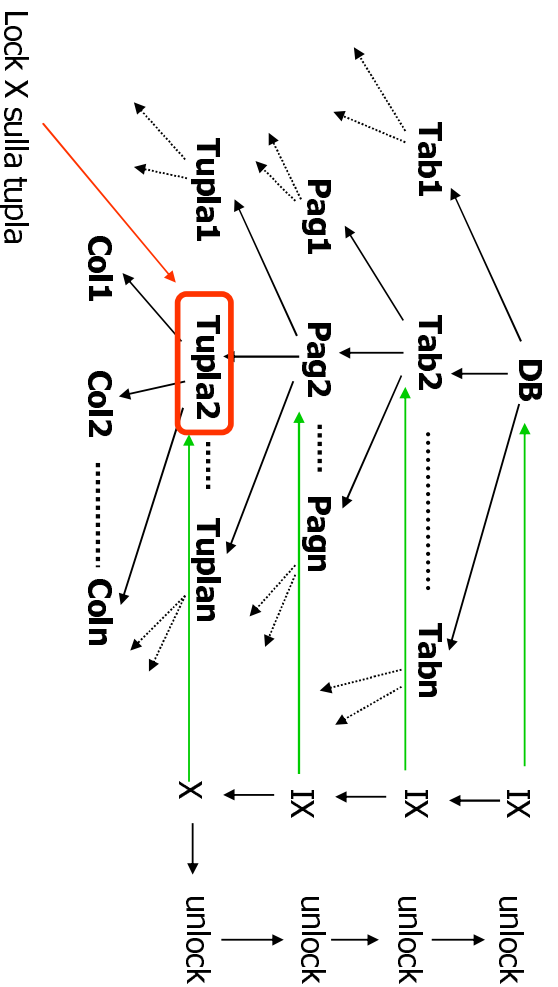
- E' possibile specificare i lock a livelli diversi (**granularità dei lock**)
 - tabelle, pagine, tuple, campi di singole tuple



Primitive per il lock gerarchico

- Si aggiungono lock specifici oltre ai lock S e X
 - **IS : Intentional Shared lock**
Esprime l'intenzione di bloccare in modo condiviso uno dei nodi discendenti del nodo corrente
 - **IX : Intentional eXclusive lock**
Esprime l'intenzione di bloccare in modo esclusivo uno dei nodi discendenti del nodo corrente
 - **SIX : Shared Intentional-eXclusive lock**
Blocca il nodo corrente ed esprime l'intenzione di bloccare in modo esclusivo uno dei nodi discendenti del nodo corrente

Lock gerarchico





Protocollo di lock gerarchico

- Si richiedono i lock a partire dalla radice scendendo lungo l'albero
- Si rilasciano i lock a partire dal nodo verso la radice
- Per richiedere un lock S o IS su un nodo si deve possedere un lock IS o IX sul padre
- Per poter richiedere un lock IX, X o SIX su un nodo si deve già possedere un lock SIX o IX sul padre
- E' definita la tabella di compatibilità per stabilire se accettare la richiesta di lock



Tabella di compatibilità

Richiesta	Stato risorsa				
	IS	IX	S	SIX	X
IS	Si	Si	Si	Si	No
IX	Si	Si	No	No	No
S	Si	No	Si	No	No
SIX	Si	No	No	No	No
X	No	No	No	No	No

Lock e B-tree

Problema quando si usano B-tree (Lock di B-tree)

- si parte dalla radice acquisendo i lock di tutti i nodi incontrati fino ai dati che vogliamo modificare/leggere
- il locking a due fasi prevede che un lock sia rilasciato solo dopo il commit
- quindi la root **rimane bloccata**
 - con gli inserimenti tutto l'albero rimane completamente bloccato (concorrenza inesistente)

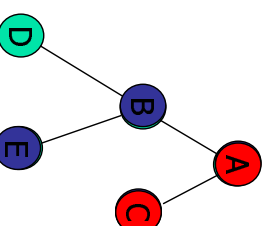
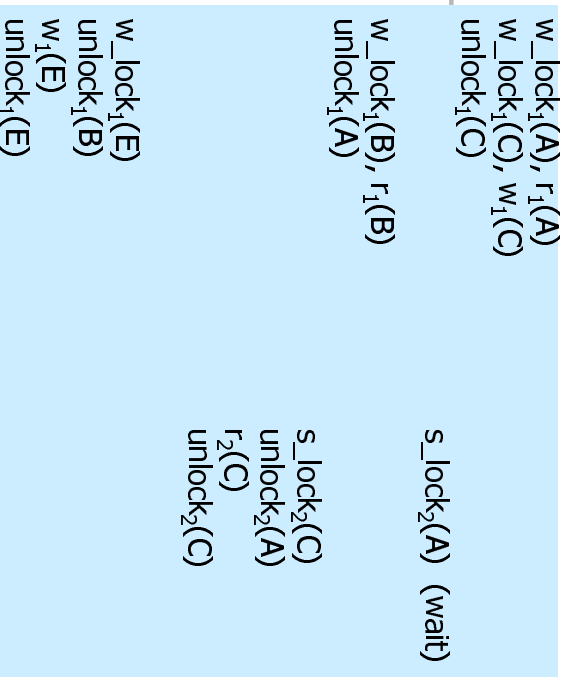
Soluzione

- definire un nuovo algoritmo di lock per gli alberi che
 - rilasci i lock non appena possibile
 - garantisca comunque la serializzabilit 

Lock e B tree II

Algoritmo

- Inizialmente, la transizione acquisisce un **lock sulla radice**
- Successivamente, puo' essere acquisito un lock su un qualsiasi nodo solo se la transizione possiede un lock sul padre
- Un lock puo' essere rilasciato in qualsiasi momento
- Un lock rilasciato non puo' essere riacquisito



Lock e B tree III

Osservazioni

- Un lock condiviso su un nodo **n** può essere rilasciato
 - **non appena** si acquisisce il lock sul figlio **f**
- Un lock esclusivo su un nodo **n** può essere rilasciato
 - quando si acquisisce il lock sul figlio **f**
 - se **non ci sono rischi che la modifica si propaghi**
 - modifica di inserimento ed **f** e' pieno
 - modifica di eliminazione ed **f** e' al limite della minima occupazione

Serializzabilita'

- Si può dimostrare che usando il protocollo descritto si generano schedule conflict-serIALIZZABILI
 - Le transizioni possono essere ordinate sulla base all'ordine di acquisizione del lock sulla radice

Deadlocks

- E' possibile che si verifichi uno stallo (**deadlock**)

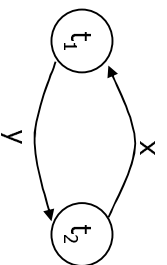
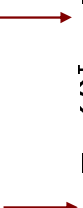
- si verifica in caso di **schedule non serializzabile secondo 2PL**

$t_1 : r_1(x) w_1(y)$

$t_2 : r_2(y) w_2(x)$

- Possibile schedule con 2PL

$r_{lock_1}(x) \ r_{lock_2}(y) \ r_1(x) \ r_2(y) \ w_{lock_1}(y) \ w_{lock_2}(x)$



t_1 si blocca aspettando
che si liberi y bloccato
da t_2

t_2 si blocca aspettando
che si liberi x bloccato
da t_1



Deadlocks II

La probabilità di un deadlock

Numero transazioni: t

Numero di oggetti disponibili: n

Numero medio di oggetti bloccati per transazione: m

La probabilità che una transazione T_1 stia attendendo una risorsa bloccata da un'altra transazione T_2 è m/n

La probabilità di un deadlock causato da T_1 e T_2 (deadlock di lunghezza due) è m^2/n^2

La probabilità che esista un deadlock di lunghezza due è $O(t^2m^2/n^2)$

- Poiché normalmente $n \gg mt$, la probabilità di avere un deadlock è bassa
- Si tratta di una sottostima a causa della **località** delle transazioni
- In pratica, i **deadlock** sono rari ma possibili



Soluzioni al deadlock

Tre soluzioni

- **Timeout**

- Le transazioni rimangono in attesa di una risorsa per un tempo prefissato
- Una risorsa in deadlock viene comunque liberata dopo il timeout
- Rimane difficile determinare il timeout ottimale
- Il timeout è comunque utile (una transazione blocca una risorsa a lungo)

- **Prevenzione (deadlock prevention)**

- Se una transazione può causare un deadlock, allora viene uccisa
- si fa in modo da non generare mai deadlock

- **Rilevamento (deadlock detection)**

- Si controlla periodicamente il sistema alla ricerca di situazioni di stallo e si uccide una delle transazioni coinvolte nel deadlock

Rilevamento e prevenzione attraverso grafo dei conflitti

Strategia

- Si costruisce il grafo dei conflitti
- Se è ciclico allora c'è un deadlock

Osservazioni

- Il grafo può essere aggiornato tutte le volte che una transazione richiede/rilascia una risorsa
 - in questo caso si tratta di **prevenzione dei deadlock**
- Il grafo può essere aggiornato in corrispondenza dei checkpoint o a scadenze predefinite
 - in questo caso si tratta di **rilevazione dei deadlock**
 - la transazione da uccidere può essere scelta in base ad un timestamp o scegliendo quella che ha fatto meno lavoro (accessi)

Prevenzione attraverso ordinamento

La strategia

- Si usa 2PL
- Si ordinano gli oggetti del database
 - es. in base al loro indirizzo
- Una transazione deve richiedere le risorse rispettando l'ordinamento
 - es. se una transazione richiede in sequenza R_1, R_2, \dots, R_n , deve essere verificato che $R_1 < R_2 < \dots < R_n$

Osservazioni

- In questo modo **non si verificano stalli**
- spesso non è possibile conoscere in anticipo quali risorse serviranno

T_1 ha bloccato le risorse R_1, R_2, \dots, R_n T_2 ha bloccato le risorse S_1, S_2, \dots, S_m
 T_2 è in attesa della risorsa R_i di T_1

T_1 non può essere in attesa di S_j di T_2 altrimenti $R_i \leq R_n \leq S_j \leq S_n < R_i$

Un esempio

- Si suppone che gli oggetti A,B,C,D siano ordinati alfabeticamente
- T1: r(A), w(B)
- T2: r(C), w(A)
- T3: r(B), w(C)
- T4: r(D), w(A)

l=lock
u=unlock

T1	T2	T3	T4
l(A), r(A)			
	l(A) (wait)		
		l(B), r(B)	
			l(A) (wait)
		l(C), w(C)	
		u(B), u(C)	
l(B), w(B)			
u(A), u(B)			
	l(A), l(C)		
	r(C), w(A)		
	u(C), u(A)		
			l(A), l(D)
			r(D), w(A)
			u(D), u(A)

Prevenzione attraverso timestamp

- Ad ogni transazione si assegna un **timestamp** (inizio della transazione)
- Quando una transazione T richiede una risorsa bloccata da S si confrontano i due timestamp h_t e h_s
- **Politica non interrompente**
 - Se $h_t < h_s$, T può attendere il rilascio della risorsa
 - Se $h_t > h_s$, T viene uccisa
- **Politica interrompente**
 - Se $h_t < h_s$, S viene forzata a rilasciare la risorsa
 - Se $h_t > h_s$, T può attendere il rilascio della risorsa
- Le transazioni uccise sono rilanciate con lo stesso timestamp che avevano all'inizio per evitare problemi di **starvation**

Esempi

T1: $r(A), w(B)$; T2: $r(C), w(A)$; T3: $r(B), w(C)$; T4: $r(D), w(A)$

T1	T2	T3	T4
$I(A), r(A)$			
	$I(A)$ (uccisa)		
		$I(B), r(B)$	
			$I(A)$ (uccisa)
		$I(C), w(C)$	
		$u(B), u(C)$	
$I(B), w(B)$			
$u(A), u(B)$			
			$I(A), I(D)$
	$I(A)$ (wait)		
			$r(D), w(A)$
			$u(D), u(A)$
	$I(A), I(C)$		
	$r(C), w(A)$		
	$u(C), u(A)$		

Politica non interrompente

T1	T2	T3	T4
$I(A), r(A)$			
	$I(A)$ (wait)		
		$I(B), r(B)$	
			$I(A)$ (wait)
		(uccisa)	
$I(B), w(B)$			
$u(A), u(B)$			
	$I(A), I(C)$		
	$r(C), w(A)$		
	$u(C), u(A)$		
			$I(A), I(D)$
			$r(D), w(A)$
			$u(D), u(A)$
		$I(B), r(B)$	
		$I(C), w(C)$	
		$u(B), u(C)$	

Politica interrompente

Prevenzione attraverso timestamp II

Perche' funziona ?

- Si supponga che esista un ciclo $T_1, T_2, \dots, T_n, T_1$ nel grafo dei conflitti
- Con la politica non interrompente le transizioni piu' vecchie aspettano quelle piu' giovani: $h_1 < h_2 < \dots < h_n < h_1$
- Con la politica interrompente le transizioni piu' giovani aspettano quelle piu' vecchie: $h_1 > h_2 > \dots > h_n > h_1$

Vantaggi e svantaggi (assumendo che molti lock siano acquisiti all'inizio)

- Politica non interrompente
 - Si uccide le transazioni che hanno fatto meno lavoro
- Politica interrompente
 - Si uccide meno transazioni



Un paragone fra i meccanismi per risolvere i deadlock

Il grafo dei conflitti

- E' il metodo **comunemente** usato dai DBMS commerciali
- riduce al minimo i roll back
- è dispendioso e complicato: occorre mantenere il grafo dei conflitti

Prevenzione attraverso timestamp

- può succedere che transazioni che non causeranno conflitti vengano uccise e devano essere rilanciate
- ha un ruolo più importante in un ambiente distribuito, dove il mantenimento del grafo dei conflitti è più gravoso



Riassumendo

- La teoria suggerisce le proprietà che uno scheduler deve avere perché non possieda anomalie: deve essere serializzabile
- La view-serializzabilità fornisce **schedule più efficienti**, ma implicerebbe uno **scheduler inefficiente**
- 2PL permette di realizzare uno **scheduler efficiente**
- Con i **lock** si implementa 2PL, quindi si garantiscono **schedule senza anomalie** (isolamento delle transazioni)
- L'uso dei lock può introdurre **deadlock** (si verificano quando lo **schedule** è non serializzabile secondo 2PL)
- Occorre adottare strategie di **prevenzione/riconoscimento dei deadlock**

Altri approcci al controllo della concorrenza

Il meccanismo dei lock assume

- se non si controlla le transazioni si verificano anomalie

Gli **approcci ottimistici** assumono

- tutto andrà bene, altrimenti interveniamo
- l'intervento consiste nell'uccidere la transazione colpevole dell'anomalia
- **basati su timestamp**
 - si verifica che lo schedule sia equivalente ad uno seriale usando dei timestamp per le transazione e le operazioni di lettura e scrittura
- **basati su validazione**
 - si esaminano i timestamp prima che la transazione faccia il commit

Controllo della concorrenza basato su timestamp

Si assegnano

- per ogni transizione T un timestamp **$TS(T)$**
- per ogni oggetto X
 1. Il maggiore timestamp fra quelli delle transazioni che hanno letto X: **$RT(X)$ (tempo di lettura)**
 2. Il maggiore timestamp fra quelli delle transazioni che hanno scritto X: **$WT(X)$ (tempo di scrittura)**
 3. Un booleano che è vero se la transazione in (2) ha fatto commit: **$c(X)$ (bit di commit)**

L'idea di base

- $TS(T)$, $RT(X)$, $WT(X)$ servono a riconoscere gli schedule non serializzabili
- $c(X)$ serve a riconoscere le letture sporche

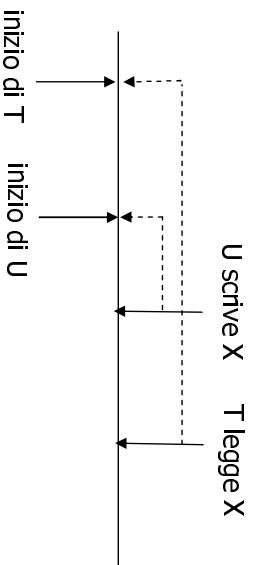
Comportamenti fisicamente non realizzabili

Si assume che tutta la transazione abbia luogo **all'istante iniziale**

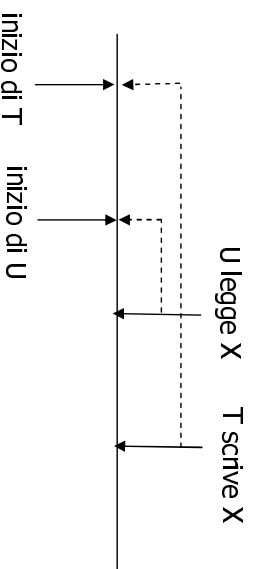
I comportamenti **fisicamente non realizzabili**

- sono gli schedule che non si comportano come uno schedule seriale
- lo schedule seriale è quello in cui le transazioni vengono eseguite all'istante $TS(T)$

Lettura in ritardo

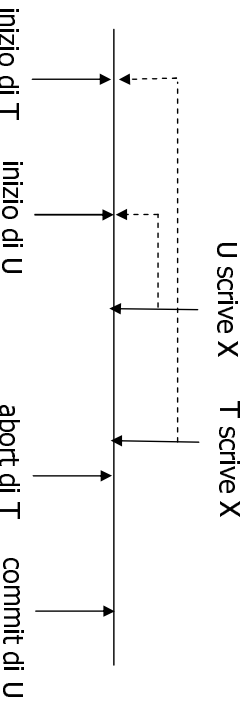
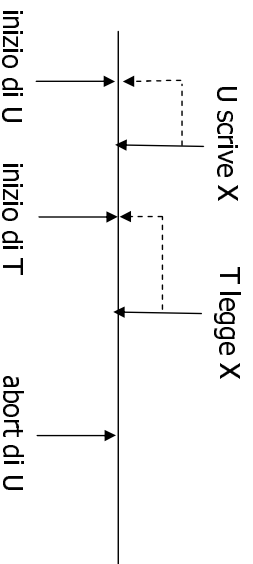


Scrittura in ritardo



Lecture sporche

Sono problemi dovuti alla presenza di ABORT in alcune transazioni

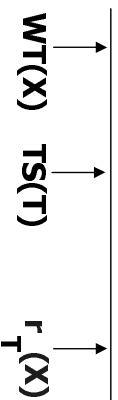


Una scrittura seguita da un'altra scrittura potrebbe essere non eseguita. Se la seconda scrittura viene abortita questo genera una anomalia.

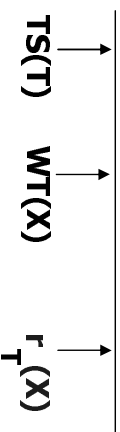
L'algoritmo

Se lo scheduler riceve una richiesta di lettura $r_T(X)$

- se $TS(T) \geq WT(X)$ la lettura è fisicamente realizzabile
 - se $c(X)$ è vero, esegui la richiesta. Aggiorna $RT(X)$
 - se $c(X)$ è falso, ritarda T fino a quando la transazione che ha scritto X abortisce o fa commit



- se $TS(T) < WT(X)$ la lettura non è fisicamente realizzabile
 - Abortisci e rilancia T



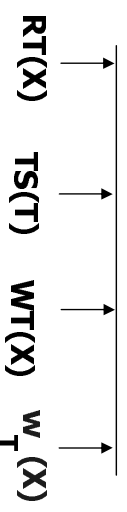
L'algoritmo II

Se lo scheduler riceve una richiesta di scrittura $w_T(X)$

- se $TS(T) \geq RT(X)$ e $TS(T) \geq WT(X)$ la scrittura è fisicamente realizzabile

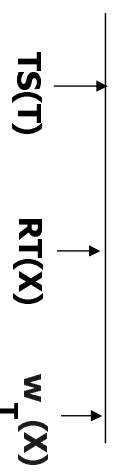


- se $TS(T) \geq RT(X)$ e $TS(T) < WT(X)$ la scrittura è fisicamente realizzabile ma X contiene un nuovo valore scritto da una transazione U



- se $c(X)$ è vero, non fare niente
- se $c(X)$ è falso, allora occorre uccidere la transazione

- se $TS(T) < RT(X)$ la scrittura non è fisicamente realizzabile



- Abortisci e rilancia T

L'algoritmo III

Se lo scheduler riceve una richiesta commit da una transazione T

- per tutti gli oggetti X modificati da T
 - assegna $c(X)=true$
 - se ci sono transazioni in attesa, attivale

Se lo scheduler riceve una richiesta abort da una transazione T

- le transazioni in attesa devono ripetere il loro tentativo di scrittura e lettura e vedere cosa succede

Un esempio

T_1	T_2	T_3	A	B	C
200	150	175	$RT=0$ $WT=0$ $C=true$	$RT=0$ $WT=0$ $C=true$	$RT=0$ $WT=0$ $C=true$
$r(B)$			$RT=200$		
	$r(A)$		$RT=150$		
		$r(C)$			$RT=175$
$w(B)$				$WT=200$ $C=false$	
$w(A)$			$WT=200$ $C=false$		
Commit			$C=true$	$C=true$	
	$w(C) \rightarrow SA$				
		$w(A)$	Commit		

oggetti

Abort forzato dallo scheduler

Si va avanti senza fare niente

Confronto fra locking e controllo basato su timestamp

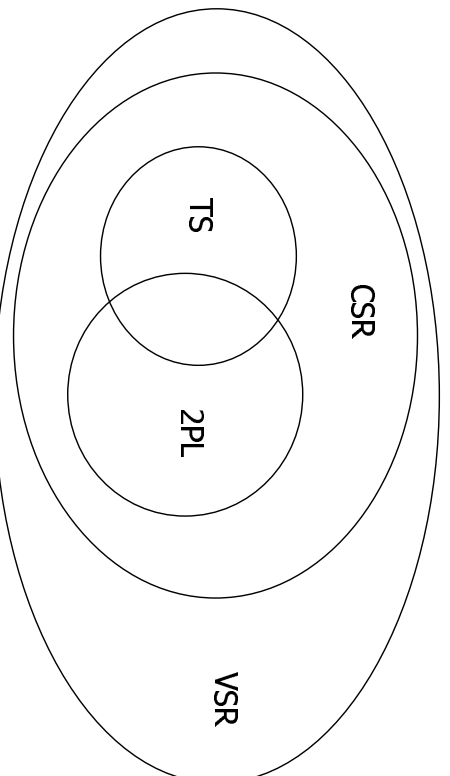
Pro e contro

- Il controllo basato su timestamp
 - è migliore nel caso la maggior parte delle transazioni sia in **sola lettura**
 - se ci sono molte transazioni in scrittura provoca **numerosi rollback**
- Il controllo basato su locking
 - spesso **ritarda** senza motivo l'esecuzione delle transazioni
 - è adottato dalla **maggior parte dei DBMS** commerciali
- Una soluzione **intermedia** in sistemi commerciali
 - dividere le transazioni fra quelle sola lettura e lettura e scrittura
 - quelle in sola lettura usano il controllo basato su timestamp, le altre il controllo basato su locking

Confronto fra locking e controllo basato su timestamp II

Da un punto di vista teorico

- Il controllo basato su concorrenza genera una classe di schedule TS contenuti in CSR
- 2PL e TS hanno un'intersezione, ma non vale nessuna inclusione





Controllo di concorrenza basato su validazione

Controllo **basato su validazione**

- E' controllo di concorrenza **ottimistico**
- Lo scheduler mantiene una lista delle operazioni di ciascuna transazione
- Subito prima di procedere al commit verifica che non esistano comportamenti non realizzabili fisicamente

L'algoritmo prevede **tre fasi** per ogni transazione

- **Lettura**: Le transazioni leggono dal database e scrivono nel loro spazio di lavoro i loro risultati
- **Validazione**: Lo scheduler valida le transazioni comparando le operazioni fatte con quelle delle altre transazioni. Se la validazione fallisce la transazione viene abortita.
- **Scrittura**: la transazione scrive i dati sul database



La validazione

Per la validazione lo scheduler deve memorizzare **tre insieme**

- **START**: l'insieme delle transazioni che sono iniziate, ma non hanno terminato la validazione
 - **VAL**: l'insieme delle transazioni che sono state validate ma che non hanno concluso la scrittura
 - **FIN**: l'insieme delle transazioni che hanno finito la scrittura
- e **tre timestamp** per ogni transazione T
- **START(T)**: il momento in cui T è iniziata
 - **VAL(T)**: il momento in cui T è stato validato (**indica anche l'ordine seriale assunto**)
 - **FIN(T)**: il momento in cui T ha finito

I tre valori sono necessari fino a quando esiste U per la quale $START(U) \leq FIN(T)$

La validazione

Inoltre per ogni transizione T si memorizza

- $RS(T)$: l'insieme di oggetti letti da T
- $WS(T)$: l'insieme di oggetti scritti da T

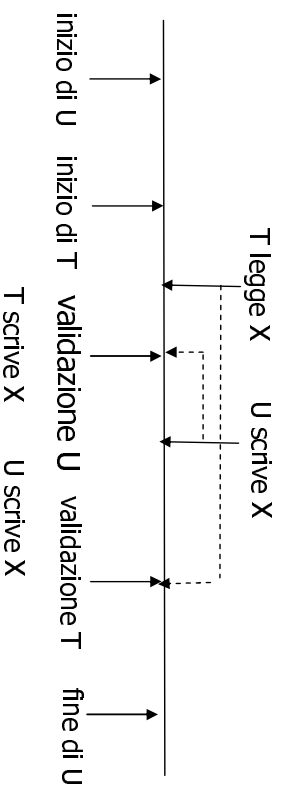
La validazione II

Si assume che tutta la transazione abbia luogo al momento della validazione

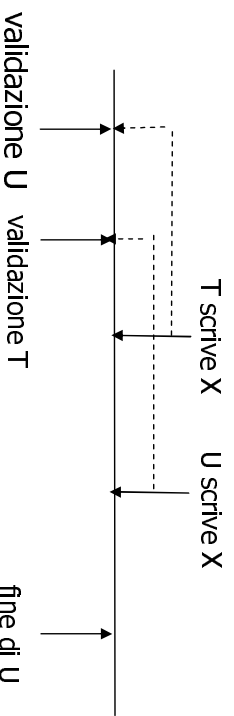
Per la validazione di T si controlla che

- $RS(T) \cap WS(U) = \emptyset$, per ogni U per il quale $FIN(U) > START(T)$
- $WS(T) \cap WS(U) = \emptyset$, per ogni U per il quale $FIN(U) > VAL(T)$

Lettura/scrittura
T ha letto il valore di X
prima che U finisse di
scrivere



Scrittura/scrittura
T potrebbe scrivere
prima di U



Un esempio

Validazione di U

- **ok**: nessun'altra transazione validata

Validazione di T

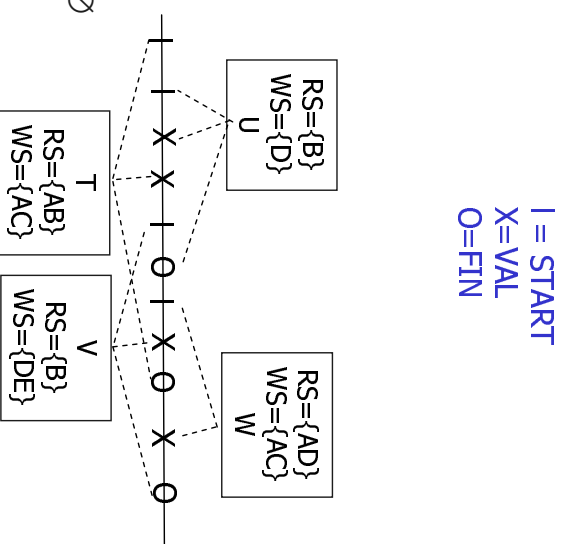
- **ok**: $WS(T) \cap WS(U) = \emptyset$, $RS(T) \cap WS(U) = \emptyset$

Validazione di V

- **ok**: $RS(V) \cap WS(U) = \emptyset$
 $WS(V) \cap WS(T) = \emptyset$, $RS(V) \cap WS(T) = \emptyset$

Validazione di W

- **no**: $RS(W) \cap WS(V) = \{D\}$, $WS(W) \cap WS(V) = \emptyset$
 $RS(W) \cap WS(T) = \{A\}$



Confronto fra i metodi di controllo della concorrenza

Spazio occupato dai i tre metodi è simile

- **Locking**:
 - la tabella di lock e' proporzionale al numero di richieste di lock in corso
- **Timestamp**:
 - 2 timestamp per ogni oggetto acceduto dalle transazioni e 1 timestamp per ogni transazione
 - Non è sufficiente tenere conto solo delle transazioni in corso, ma servono anche quelle recenti



Confronto fra i metodi di controllo della concorrenza

Spazio occupato dai i tre metodi è simile

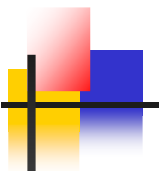
- **Locking:**
 - la tabella di lock e' proporzionale al numero di richieste di lock in corso
- **Timestamp:**
 - 2 timestamp per ogni oggetto acceduto dalle transazioni e 1 timestamp per ogni transazione
 - Non è sufficiente tenere conto solo delle transazioni in corso, ma servono anche quelle recenti
- **Validazione:**
 - spazio per gli insiemi RS, WS di oggetti acceduti dalle transazioni e 3 timestamp per ogni transazione
 - pero' deve tenersi in un area locale le modifiche fatte da ogni transazione



Confronto fra i metodi di controllo della concorrenza II

Osservazioni

- Il locking ritarda le transazioni, ma evita i rollback
- Il timestamp e la validazione sono migliori in caso di bassa interferenza fra le transazioni
- quando si deve uccidere una transazione
 - il timestamp identifica il problema prima
 - la validazione aspetta il completamento delle transazioni
- nel caso di transazioni che possono causare un abort
 - la validazione permette di continuare l'esecuzione
 - il timestamp la blocca



Basi di dati distribuite



Basi di dati distribuite

Una **base di dati distribuita**

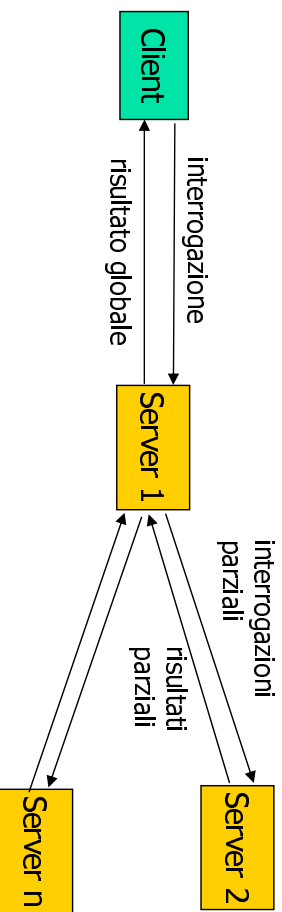
- i dati sono distribuiti su server (DBMS) diversi
- le transazioni coinvolgono server diversi
- i DBMS possono essere **omogenei** o **eterogenei**
- può essere locale o geografica

L'esecuzione di una **transazione distribuita**

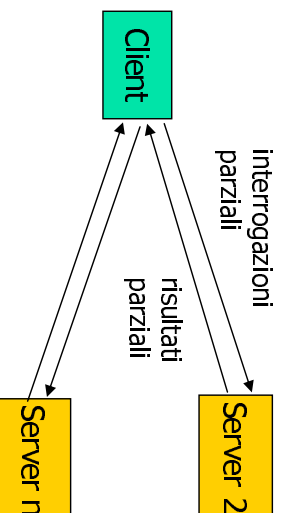
- la base di dati può presentare diversi livelli di **trasparenza**
 - **un estremo**: l'utente conosce un solo server a cui invia richieste SQL come se i dati si trovassero su quel server
 - **l'altro estremo**: l'utente deve conoscere l'esatta locazione dei dati e i linguaggi usati dai vari server

Interazione client – DBMS

Schema di interazione 1



Schema di interazione 2



Livelli di trasparenza: un esempio

FORNITORE(codice, Nome, Città)

Nella sede di Milano: $\sigma_{Città="milano"}(FORNITORE)$

Nella sede di Roma: $\sigma_{Città="roma"}(FORNITORE)$

Esempio di frammentazione orizzontale

Trasparenza completa

```
SELECT nome INTO :n
FROM fornitore
WHERE codice=:c
```

Trasparenza di linguaggio

```
SELECT nome INTO :n
FROM fornitore1@milano.ditta.it
WHERE codice=:c

if :empty then
SELECT nome INTO :n
FROM fornitore2@roma.ditta.it
WHERE codice=:c
```

Nessuna trasparenza

1. Apri una connessione con Milano
2. Invia un'interrogazione nel linguaggio del server di Milano
3. Ricevi la risposta
4. Apri una connessione con Roma
5. Invia un'interrogazione nel linguaggio del server di Roma
6. Ricevi la risposta

Trasparenza di allocazione

```
SELECT nome INTO :n
FROM fornitore1
WHERE codice=:c

if :empty then
SELECT nome INTO :n
FROM fornitore2
WHERE codice=:c
```


Classificazione delle transazioni

Richieste remote

- sono transazioni di **sola lettura** indirizzate ad un solo DBMS remoto

Transazioni remote

- sono transazioni costituite da un numero qualsiasi di comandi SQL indirizzate ad **un solo DBMS** remoto

Richieste distribuite

- sono transazioni **rivolte a piu' DBMS**, ma ogni comando SQL fa riferimento ad un solo DBMS

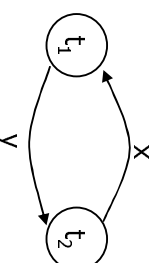
Transazioni distribuite

- sono transazioni rivolte a piu' DBMS, dove ogni comando SQL puo' far riferimento a **dati presenti su DBMS diversi**

Controllo di concorrenza in ambito distribuito

Teoria

- una **transazione distribuita** è rappresentata da **un insieme di sottotransazioni**, ognuna relativa a un nodo
- es:
 - $t_1: r_{11}(X) \ w_{11}(X) \ r_{12}(Y) \ w_{12}(Y)$
 - $t_2: r_{22}(Y) \ w_{22}(Y) \ r_{21}(X) \ w_{21}(X)$
(il primo indice indica la transazione, il secondo indica il nodo)
- La **serializzabilita' locale** degli schedule **non garantisce quella globale**
- es: i seguenti sono due schedule relativi a t_1 e t_2 che sono **localmente seriali**, ma **non globalmente serializzabili**
 - $S_1: r_{11}(X) \ w_{11}(X) \ r_{21}(X) \ w_{21}(X)$
 - $S_2: r_{22}(Y) \ w_{22}(Y) \ r_{12}(Y) \ w_{12}(Y)$





Controllo di concorrenza in ambito distribuito II

La **serializzabilita' globale**

- Richiede l'esistenza un unico schedule globale S e schedule locali S_i
 - S deve essere seriale
 - la proiezione di S sui nodi deve produrre gli S_i

Lo schedule S è **serializzabile**

- secondo il **locking a due fasi**
 - se ogni nodo applica il metodo 2PL stretto per le operazioni di competenza
 - l'azione di commit viene fatta in maniera atomica in un istante in cui tutti i nodi detengono tutte le risorse necessarie
- secondo il metodo del **timestamp**
 - le sottotransazioni acquisiscono un unico timestamp
 - i nodi applicano localmente il controllo di concorrenza basato sul timestamp



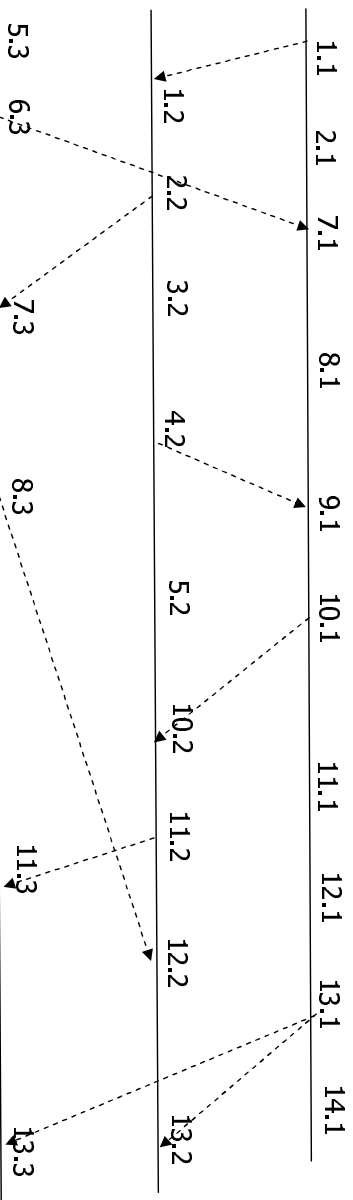
Metodo di Lamport per assegnare i timestamp

Per avere i timestamp **unici in ambito distribuito**

- Il timestamp è composto di due gruppi di cifre
 - le meno significative identificano **il nodo**
 - le piu' significative identificano **gli eventi che accadono sul nodo**
- l'indice relativo agli eventi
 - viene incrementato ad ogni evento locale
 - viene sincronizzato ad ogni scambio di messaggi
- I timestamp **rispettano l'ordinamento degli eventi sui nodi**

Metodo di Lamport per assegnare i timestamp II

Un esempio con tre nodi



Commit distribuito

Atomicita' in ambito distribuito

- tutti i nodi partecipanti in una transazione devono **fare la stessa cosa**: commit o rollback
- occorre garantire che eventuali guasti non lascino i nodi in uno stato **non coerente**
- Il protocollo piu' usato è il **commit a due fasi**

Tipi di guasti

- caduta di un nodo
- perdita di messaggi
- caduta della rete ed eventuale partizionamento



Commit a due fasi

Il commit a due fasi si svolge attraverso uno scambio di messaggi fra

- **resource manager**
i server che gestiscono di dati
- **transaction manager**
il coordinatore che ha il compito di controllare l'andamento della transazione

Consiste di due fasi

1. il transaction manager chiede ai resource manager se vogliono fare un abort o un commit e raccoglie le risposte
2. il transaction manager decide se fare abort o commit e ne informa i resource manager



Commit a due fasi II

Il protocollo prevede la scrittura **sincrona** di nuovi record sul log

Il transaction manager

- **prepare**
 - contiene la lista dei processi dei resource manager coinvolti
 - indica l'inizio del protocollo
- **global commit (global abort)**
 - esprime la decisione di fare un commit (un abort)
- **complete**
 - indica la fine del protocollo

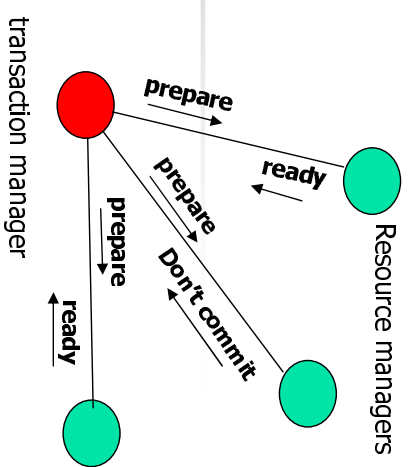
Il resource manager

- **ready**
 - indica la disponibilità a partecipare al protocollo e a fare il commit della transazione
- **don't commit**
 - indica la disponibilità a partecipare al protocollo e l'impossibilità di fare il commit

Commit a due fasi III

Fase 1 (si decide cosa fare)

- Il transaction manager
 - scrive **prepare** sul log
 - **invia un messaggio** ai resource manager
 - si mette in attesa della risposta impostando un timeout



- I resource manager

- non appena hanno finito di eseguire la loro parte di transazione scrivono **ready** sul log
- quando arriva il messaggio del transaction manager, **rispondono**
 - se sono in stato affidabile, i resource manager rispondono con **ready**
 - se non sono in stato affidabile, i resource manager rispondono con **don't commit**

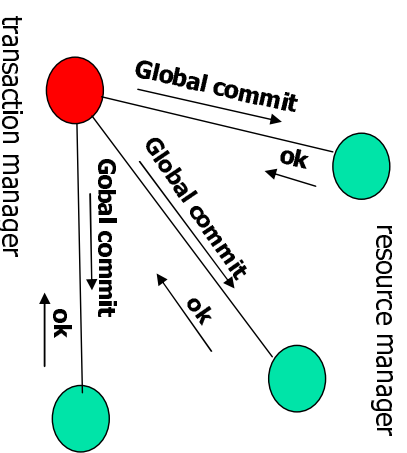
- Il transaction manager

- **colleziona** le risposte
 - se tutte le risposte sono **ready** scrive **global commit** sul log
 - se almeno una risposta è **don't commit** scrive **global abort** sul log

Commit a due fasi IV

Fase 2 (si implementa la decisione)

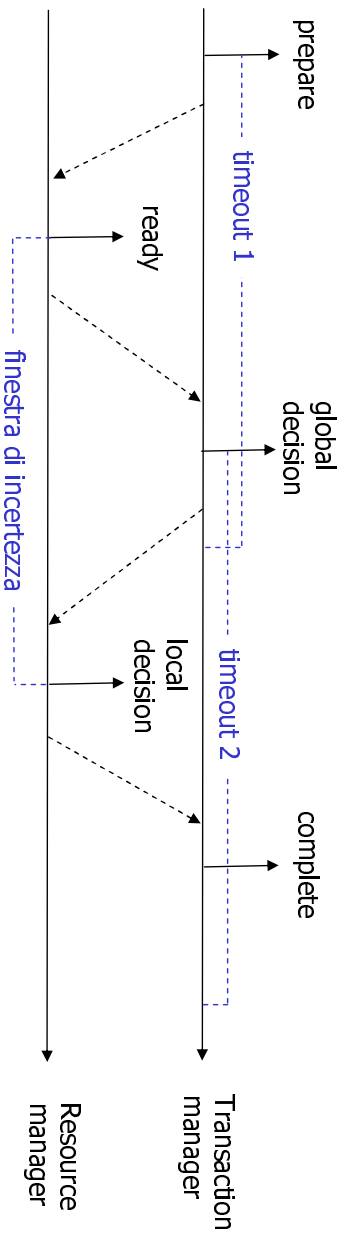
- Il transaction manager
 - comunica la decisione a tutti i resource manager
- I resource manager
 - scrivono sul log **commit** o **abort**
 - inviano un acknowledge al transaction manager
- Il transaction manager
 - quando ha ricevuto tutti gli acknowledge scrive **complete** sul log



Osservazioni

Osservazioni

- la mancanza di comunicazione fra transaction e resource manager causa
 - durante fase 1, l'abort della transazione
 - durante fase 2, la ripetizione dell'invio del messaggio
- il protocollo è progettato per ridurre al minimo la finestra di incertezza
- durante la finestra di incertezza alcune risorse rimangono bloccate



Ripristino

In caso di perdita di messaggi

- **prepare e ready**
 - scatta il timeout e si procede con un global abort
- **messaggio di decisione e acknowledge**
 - scatta il timeout e si ripete la seconda fase

In caso di caduta del resource manager, si ripristina il server e si controlla il log:

- se l'ultima scrittura di una transazione è **ready**
 - si attende che il transaction manager comunichi l'esito della transazione
 - oppure si chiede esplicitamente l'esito
- **altrimenti**
 - si procede con una normale ripartenza a caldo



Ripristino II

In caso di **caduta del transaction manager**, si ripristina il server e si controlla il log:

- se l'ultima scrittura di una transazione è **prepare**
 - si decide per un **global abort** e si procede alla seconda fase (un resource manager deve sempre rispondere a piu` messaggi uguali)
 - oppure si tenta di ripetere la prima fase
- se l'ultima scrittura di una transazione è una **global decision**
 - il transaction manager **ripete la seconda fase**
- se l'ultima scrittura di una transazione è **complete**
 - non si fa niente



Ripristino III

Cosa fare se il transaction manager **non puo' essere ripristinato** ?

- se **elegge** un nuovo transaction manager
 - es. il resource manager che ha l' IP piu' grande
 - attenzione alle **partizioni della rete**!!
- Il nuovo transaction manager contatta tutti i resource manager per decidere cosa fare
 - se **qualche nodo ha un abort o un commit**:
la decisione era gia' stata presa, si ripete la seconda fase
 - se **nessuno ha un abort o un commit**, almeno uno non ha ready:
si decide per un global abort
 - se **tutti hanno un ready**, ma nessuno un abort o un commit:
gestito manualmente da un operatore!

Estensioni del commit a due fasi

Abort presunto

- **semplifica alcune scritture dei messaggi**
- in caso di abort, i resource manager non devono rispondere al global abort e il resource manager non deve scrivere il global abort in maniera sincrona
- nel caso il transaction manager riceva una richiesta di recovery e non abbia traccia della transazione, risponde sempre con un global abort

Sola lettura

- se un resource manager effettua **solo letture**
 - invia al transaction manager un messaggio **read only**
 - il transaction manager **lo ignora** nel seguito del protocollo

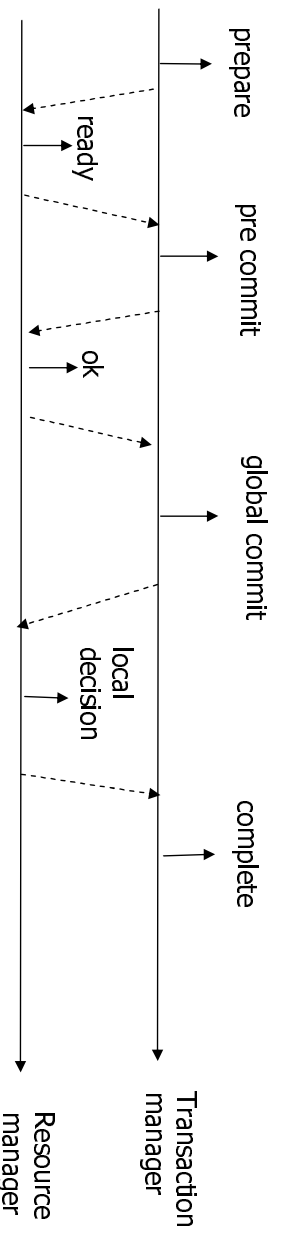
I DBMS commerciali

- adottano il commit a due fasi con abort presunto e sola lettura

Commit a tre fasi

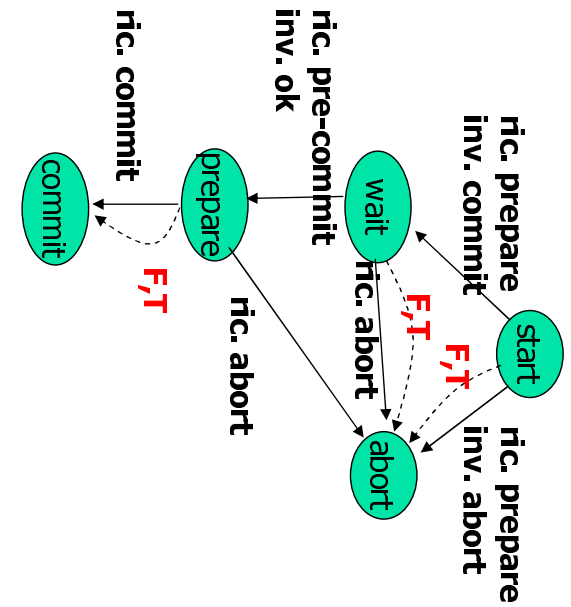
Commit a tre fasi

- include una fase intermedia in cui il transaction manager invia un pre-commit ai resource manager a cui i resource manager rispondono con un acknowledge
- riduce al minimo il tempo di blocco delle risorse: si verificano nel two phase commit nella fase di incertezza:
 - permette ai resource manager di prendere una decisione immediata in caso di fallimento o di time out

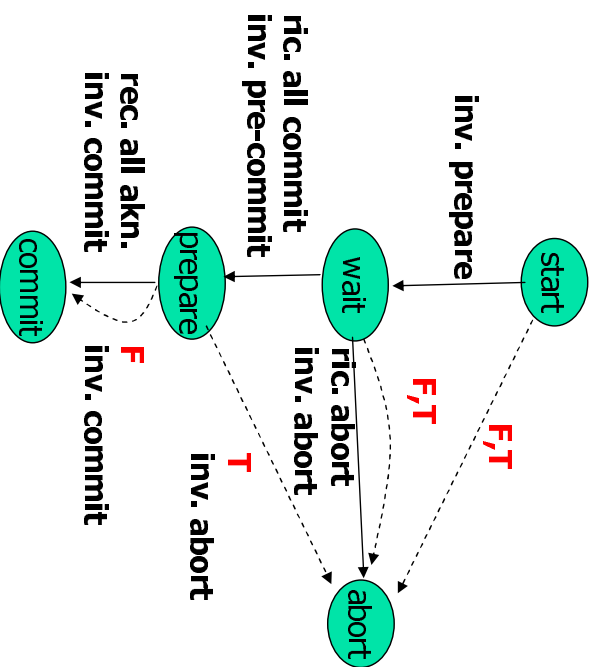


Commit a tre fasi

Resource manager



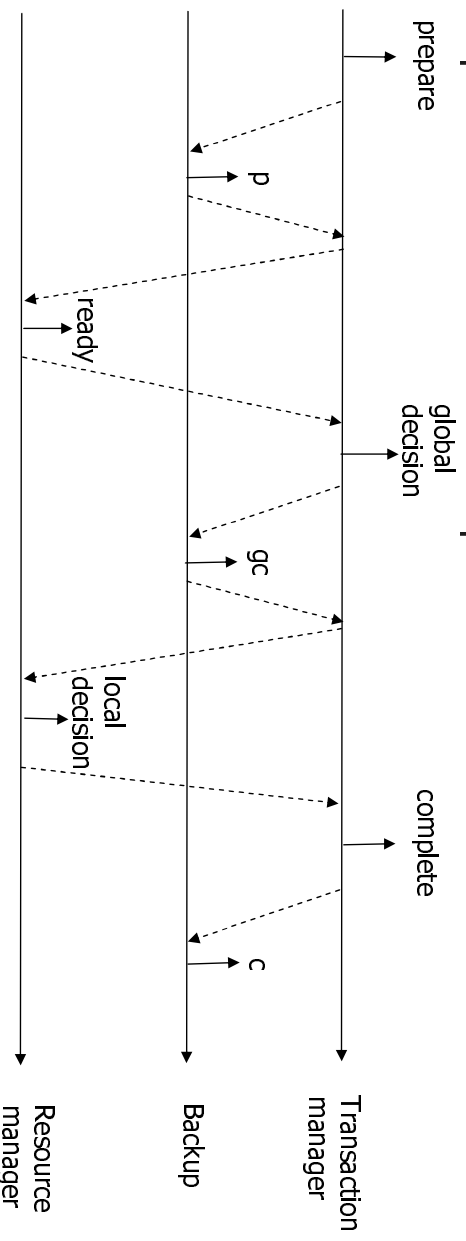
Transaction manager



Commit a quattro fasi

Commit a quattro fasi

- include un server di **backup** che ha il compito di sostituire il transaction manager in caso di caduta
- implementato da Tandem che produce architetture fault tollerant



Interoperabilità' del commit a due fasi

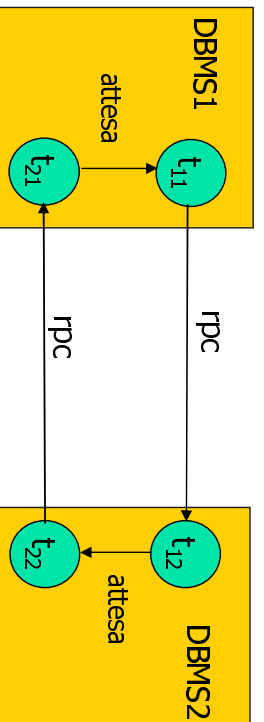
X-Open DTP (Distributed Transaction Processing)

- permette di realizzare il commit a due fasi con DBMS eterogenei
- consta di due interfacce principali, ognuna delle quali mette a disposizione una serie di procedure
 - l'interfaccia fra client e transaction manager (**TM-interface**)
tm_init, tm_exit, tm_open, tm_begin, tm_commit
 - l'interfaccia fra resource manager e transaction manager (**XA-interface**)
xa_open, xa_close, xa_start, xa_end, xa_precommit, xa_commit, xa_abort, xa_recover, xa_forget
- i resource manager sono totalmente passivi, **guidati dal transaction manager**
- il client dialoga con il transaction manager per richiedere le transazioni

Rilevazione dei deadlock distribuita

La rilevazione dei deadlock a livello di nodo non è sufficiente

- es:
 - t_{11} attende t_{12} (attivata tramite una chiamata di procedura remota)
 - t_{12} attende una risorsa controllata da t_{22}
 - t_{22} attende t_{21} (attivata tramite una chiamata di procedura remota)
 - t_{21} attende una risorsa controllata da t_{11}



Tali condizioni dai attesa saranno rappresentate come

- $E_1 \rightarrow t_{12} \rightarrow t_{22} \rightarrow E_2$
- $E_2 \rightarrow t_{21} \rightarrow t_{11} \rightarrow E_1$



Rilevazione dei deadlock distribuita II

Alcune soluzioni

- Time out
 - La soluzione piu' semplice
 - Possono essere rilevati deadlock fantasma "phantom deadlocks" a causa di ritardi sulla rete
- Un server è preposto alla rilevazione dei deadlock
 - riceve periodicamente dagli altri server i grafi locali dei conflitti
 - li unisce per decidere se esiste uno stallo
- Soluzione distribuita
 - i server si scambiano i grafi dei conflitti fino a quando uno di essi non è in grado di identificare uno stallo

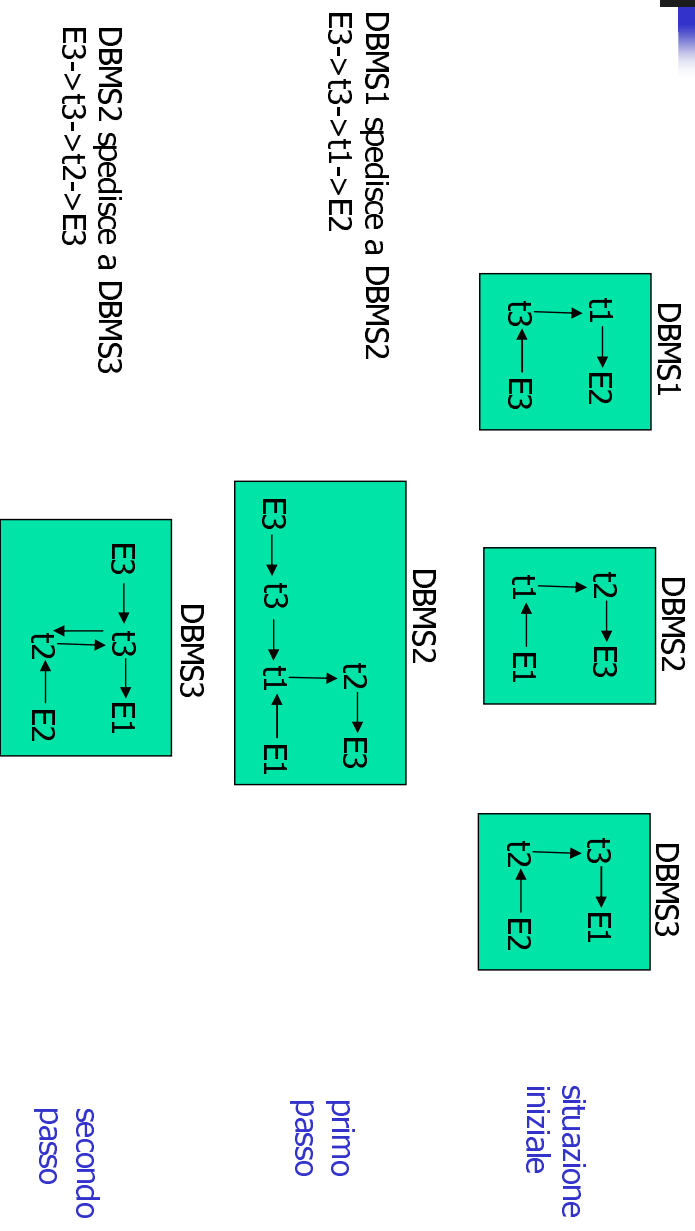


Soluzione distribuita

L'algoritmo

1. Ad ogni nodo si calcolano tutte le condizioni del tipo
 $E_{in} \rightarrow t_i \rightarrow t_j \rightarrow E_{out}$
2. Si invia al nodo out (in avanti) tutte le condizioni per cui $i > j$
(serve ad evitare che piu' nodi possano scoprire lo stesso deadlock)
3. ogni nodo svolge la ricerca di deadlock
4. si ritorna ad 1

Soluzione distribuita: un esempio



Basi di dati replicate

I dati di un archivio possono essere replicati per

- garantire la disponibilità dei dati in caso di malfunzionamento
- migliorare la velocità di accesso

La replicazione permette di creare copie fra server diversi di tutto un database o di fragmenti dei dati

- **copia verticale**
si copiano alcune colonne di una tabella, ad esempio il nome e cognome di ogni cliente
- **copia orizzontale**
si copiano alcune righe di una tabella, ad esempio tutti gli studenti lavoratori



Basi di dati replicate II

Replicazione sincrona

- I dati sono replicati in maniera sincrona, garantendo che ogni server contenga esattamente sempre la stessa informazione
- Ogni modifica ai dati viene operata su tutti le copie, magari utilizzando un commit a due fasi
- non funziona molto bene, se i server sono lontani e le transazioni distribuite sono difficili

Replicazione asincrona

- le modifiche vengono propagate in maniera asincrona
- i dati sono replicati ad intervalli regolari o in maniera continua
- i server possono contenere dati diversi, ma con il passare del tempo i dati tendono a diventare uguali



Basi di dati replicate III

Server primario e server secondari

- esiste un server primario che è proprietario di alcuni fragmenti dei dati: solo il server primario può modificare i dati che esso pubblica
- i server secondari si registrano presso il server primario: ricevano una copia dei dati e delle eventuali modifiche
- un server primario per un certo fragmento, può essere secondario per un altro

Peer-to-peer

- si permette a più server di modificare gli stessi dati
- occorre definire un meccanismo di risoluzione dei conflitti: così in cui una dato viene modificato da due server
- la risoluzione dei conflitti è e spesso basata su regole ad hoc



Basi di dati replicate IV

Nelle basi di dati replicate si fanno due tipi di copie: snapshot e transazionale

- lo snapshot consiste di una copia dei dati contenuti nel frammento
- una copia transazionale contiene le transazioni fatte su un certo frammento
- Inizialmente, il serve primario fa e distribuisce uno snapshot
- Successivamente, può distribuire le copie transazionali: in questo caso, i server secondari applicano le transizioni ai propri dati

Le copie transazionali possono essere catturate

- leggendo il log
- con dei trigger che si attivano quando viene fatta una modifica sul database



On Line Analytical Processing



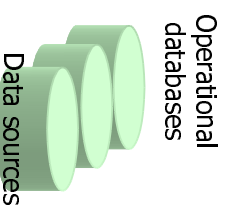
Data Warehouse

- **Data Warehouse (magazzino dati)**
 - integra in un **unico schema globale** l'informazione estratta da piu' sorgenti
 - solitamente è interrogabile, ma **non modificabile**
 - è un'**architettura per l'integrazione** di database alternativa alla replicazione ai database distribuiti
 - può essere
 - **ricostruito periodicamente** (es. tutte le notti)
 - **aggiornato periodicamente** (es. tutte le notti)
 - **aggiornato immediatamente** (es. ogni **n** transazioni)



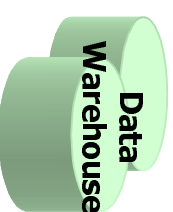
OLTP vs OLAP

- **OLTP** (On Line Transaction Processing)
 - **Gestione efficiente** dei dati in linea (molti operatori)
 - **Dati privi di errori e completi**
 - **Dati relativi allo stato attuale** dell'azienda
 - **Semplici da realizzare** e diffusi capillarmente
 - I sistemi OLTP non sono adatti all'analisi dei dati



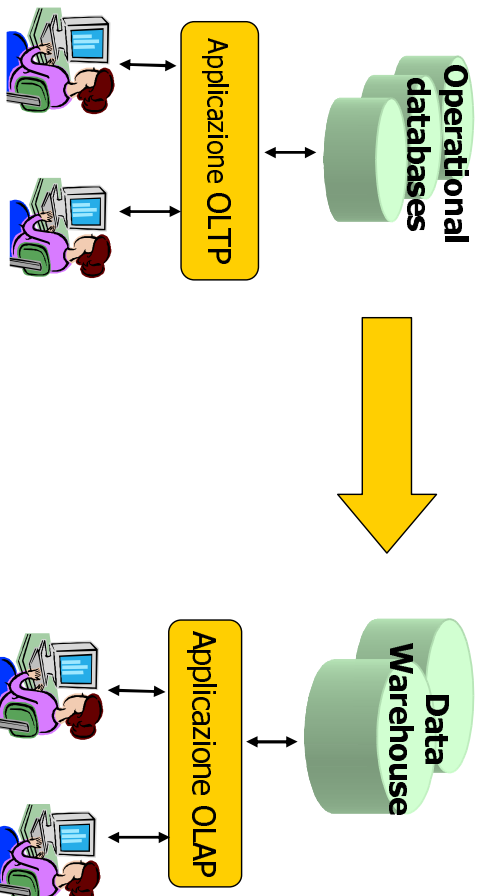
OLTP vs OLAP II

- **OLAP** (On Line Analytical Processing)
 - **Pochi utenti** dedicati all'analisi dell'andamento dell'azienda
 - I **dati storici** possono essere utili per la **pianificazione** e il **supporto alle decisioni**
 - Capire quali prodotti sono di maggiore successo
 - Stabilire l'efficacia delle promozioni sui prodotti
 - Grandi quantità di dati
 - Dati spesso scorretti o incompleti



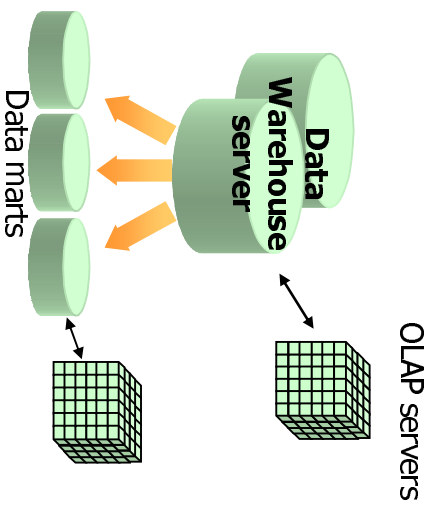
OLTP vs OLAP III

- Gli OLTP sono la fonte principale di informazioni per gli OLAP



Perché OLAP

- Definizione di un'interfaccia di analisi dei dati per utenti che svolgono attività di supporto alle decisioni
- Ottimizzazione delle operazioni di analisi invece che di gestione in linea
- Si separa l'ambiente on-line da quello di analisi
- E' l'analisi che avviene in modo interattivo on-line
- Il centro è il **magazzino dati** (data warehouse - DW)



Esempi di applicazioni OLAP

- Vendite
 - analisi e predizione delle vendite
- Marketing
 - analisi delle ricerche di mercato
 - analisi delle promozioni
 - analisi dei consumi
 - segmentazione dei mercati/clienti
- Produzione
 - Pianificazione della produzione
 - Analisi dei difetti

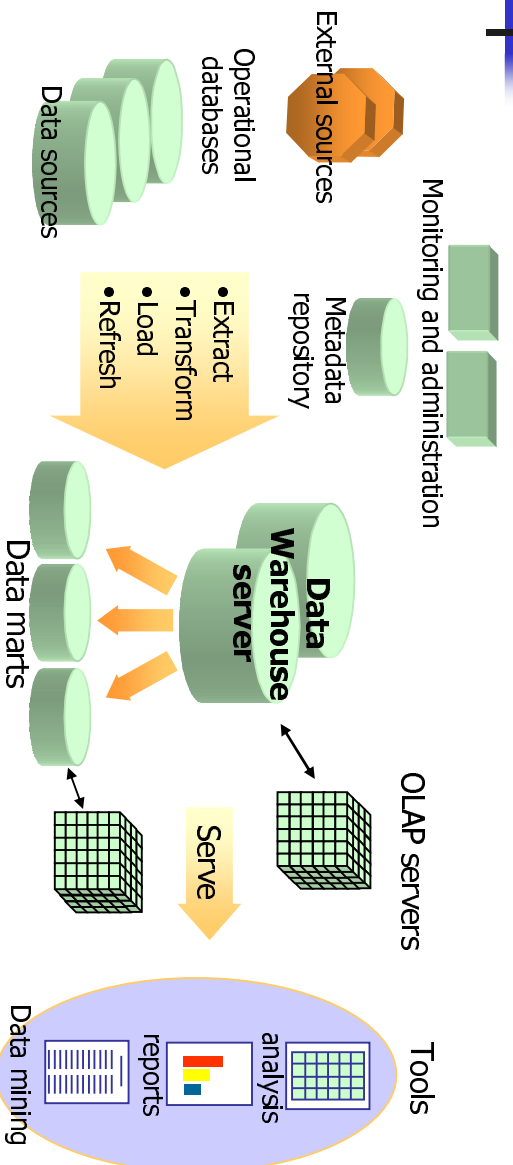
E' un investimento!

- Studio dell'International Data Corporation (IDC) 1996
 - Alti costi per l'implementazione di una efficace DW
 - Ritorno di investimento medio in 3 anni del 401%
 - 90% delle aziende con ritorno superiore al 40%
 - 25% delle aziende con più del 600% di ritorno

MA

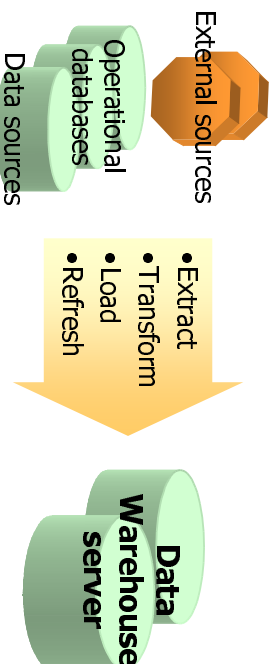
- Maggiore produttività dei *decision-makers*

On Line Analytical Processing



- OLAP → Sistemi orientati all'**elaborazione** e all'**analisi** dei dati

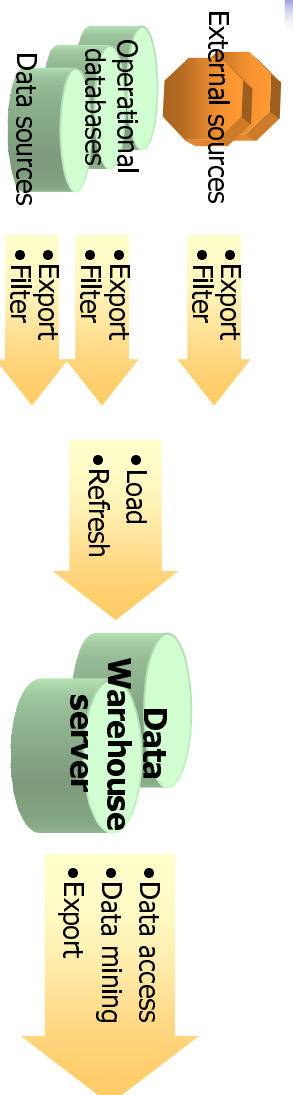
Data warehousing



I sistemi OLTP (i database operativi) sono una fonte dati

- Il **data warehouse server** trasferisce i dati dai database operativi al suo interno **integrandoli** (vista unitaria dei dati)
- I dati nella data warehouse sono di tipo **storico-temporale**
 - L'importazione dei dati è **asincrona** (disallineamento controllato dei dati) e **periodica** (riallineamento batch)
 - Occorre garantire la **qualità dei dati** nella DW

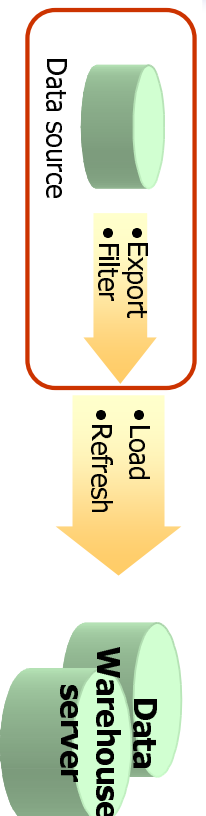
La data warehouse



Fonti dati eterogenee

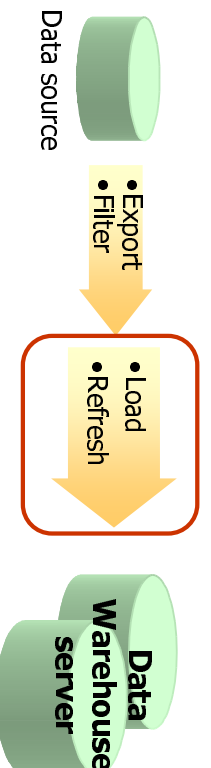
- Raccolte dati non gestite da DBMS (es. i log di un Web server)
- Dati gestiti da DBMS di vecchia generazione (**legacy systems**)
- Accesso tramite **connectors** e **filtri** dati forniti con il DW server
- Creazione/aggiornamento tramite procedure di
 - acquisizione (**load**)
 - aggiornamento (**refresh**)

Data filter & export



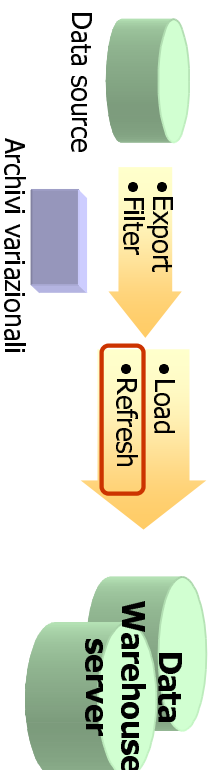
- I **data filter** controllano la correttezza dei dati prima dell'inserimento nella DW effettuando il **data cleaning**
 - eliminazione dei dati scorretti con vincoli e controlli su una o più sorgenti dati
- I **data export** sono i driver che consentono l'estrazione dei dati da una certa sorgente (DBMS, documenti, ...)
 - deve gestire un aggiornamento incrementale (basato sulle modifiche)

Data load & refresh



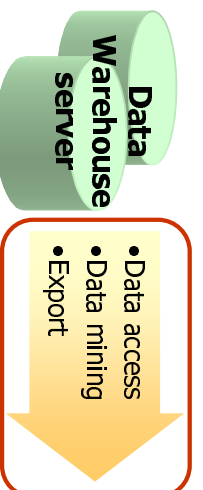
- Componente di **acquisizione dati (load)**
 - inserimento iniziale dei dati
 - costruzione delle strutture dati della DW
 - eseguita batch quando la DW non è usata
- Componente di **allineamento dati (refresh)**
 - aggiornamento incrementale della DW
 - utilizza le modifiche sulle sorgenti dati

Data refresh



- Il refresh è basato su **archivi variazionali** che registrano cancellazioni, inserimenti e modifiche
 - **Data shipping (snapshot)**
uso triggers nella sorgente e trasferimento dei dati modificati
 - **Transaction shipping (copia transazionale)**
uso dei log e trasferimento delle transazioni
- Per le cancellazioni i dati non sono eliminati nella DW per non perdere lo storico ma solo marcati

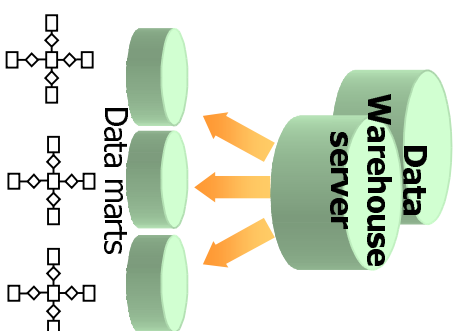
Analisi con la DW



- Componente di **accesso ai dati**
 - realizza in modo efficiente operazioni complesse di analisi
 - join fra tabelle, ordinamenti, aggregazioni -
 - operazioni come **roll up**, **drill down** e **datacube**
 - interfaccia di facile uso per gli analisti
- Componente di **data mining**
 - scoprire in modo automatico regolarità nei dati
- Componente di **esportazione dei dati** verso altre DW (es. da un DW dipartimentale ad un DW di tutta l'azienda)

I data mart

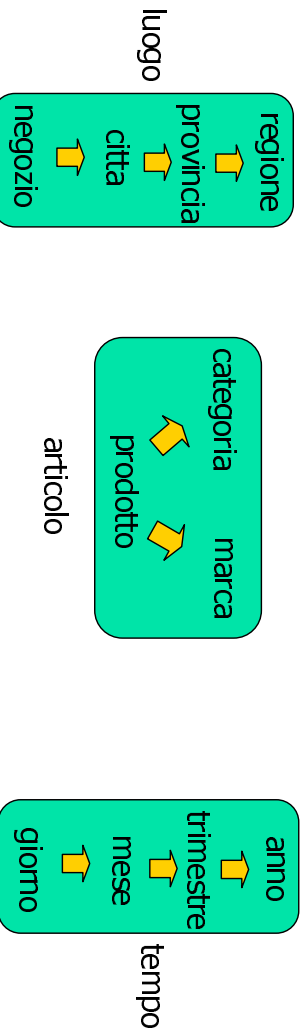
- Costruire una DW aziendale completa è complesso
- Si costruiscono schemi per sottoinsiemi semplici dei dati aziendali (**data mart**) per i quali è chiaro l'obiettivo dell'analisi
 - vendite
 - operazioni di sportello



- I dati di un data mart sono rappresentati secondo uno **schema multidimensionale** (**data cube**) e realizzato attraverso uno **schema a stella**

Il modello multidimensionale

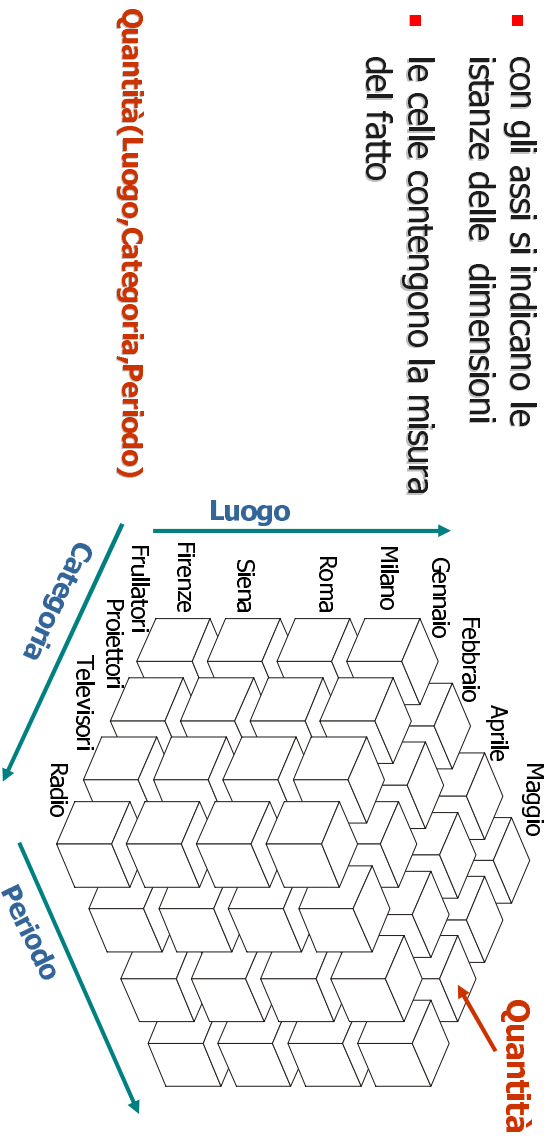
- Il modello multidimensionale è costituito da
 - **fatti**
un concetto da analizzare (es. la vendita di un prodotto)
 - **misure**
una proprietà atomica di un fatto (es. il numero delle vendite, l'incasso)
 - **dimensioni**
una prospettiva lungo la quale si analizza il fatto (es. il negozio, il mese)
- Le dimensioni sono organizzate in livelli di aggregazione



Data cube

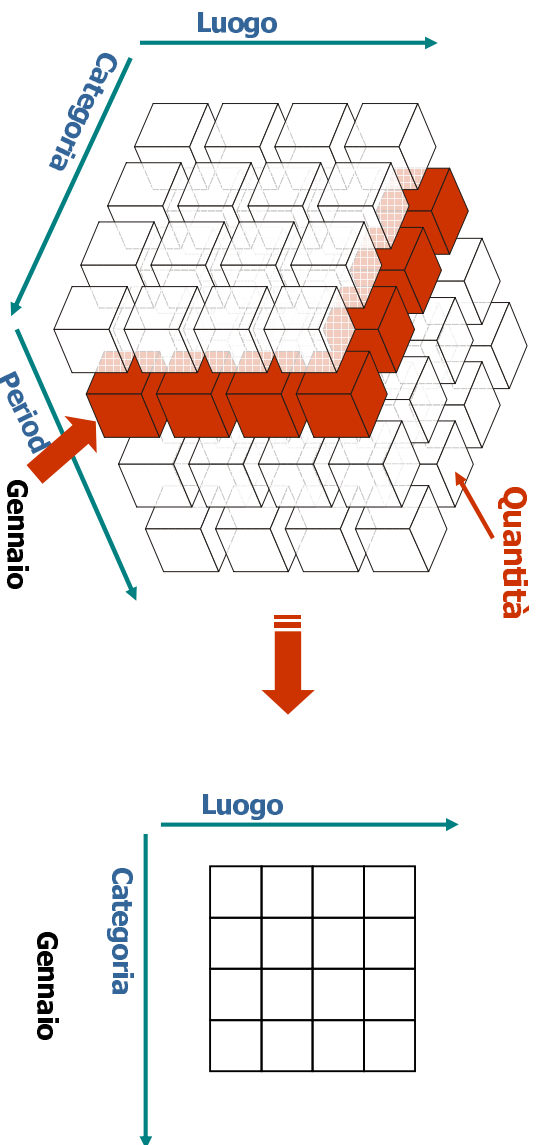
- **Data cube**

- con gli assi si indicano le istanze delle dimensioni
- le celle contengono la misura del fatto



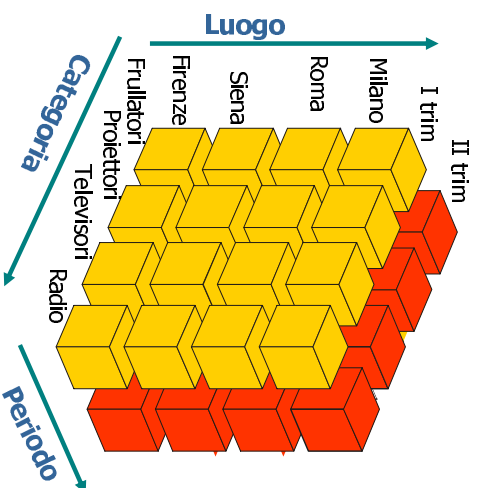
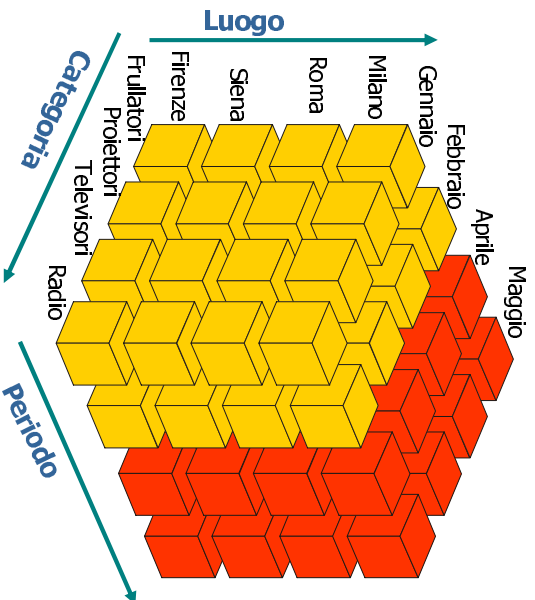
Data cube: operazioni

- Selezione di una vista (**slice-and-dice**)



Data cube: operazioni II

- Aggregazione dei dati lungo una dimensione salendo nella gerarchia (roll-up)

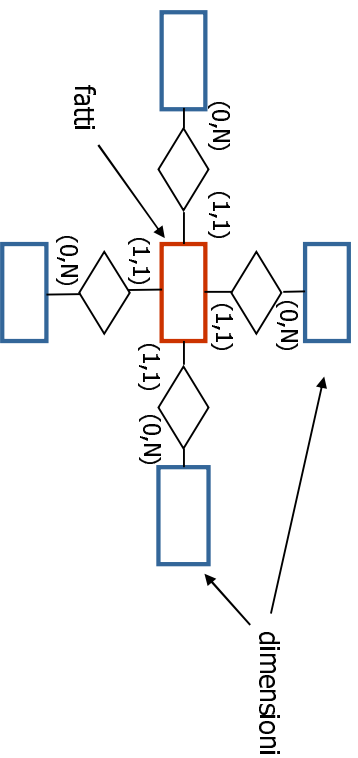


Data cube: operazioni III

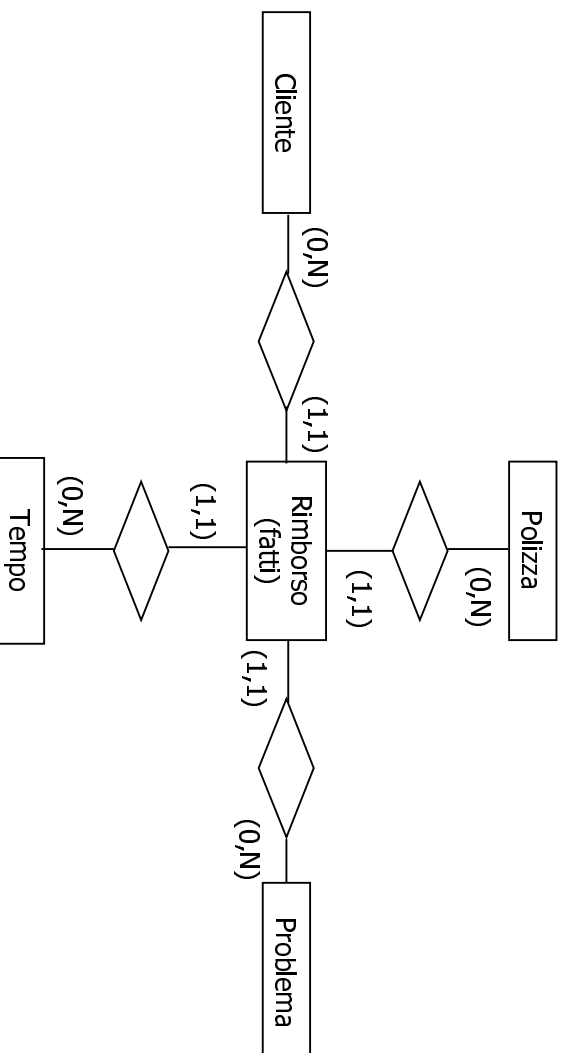
- Al limite l'operazione di roll-up elimina una dimensione
- affinché sia applicabile, occorre che la misura sia **additiva** lungo la dimensione prescelta
 - quantità e incasso sono additive rispetto al periodo
 - scorte non è additiva rispetto al periodo
- L'operazione di drill-down
 - i dati sono disaggregati e resi più dettagliati muovendo una dimensione verso il basso della gerarchia
 - è l'opposto di roll-up

Realizzazione: schema a stella

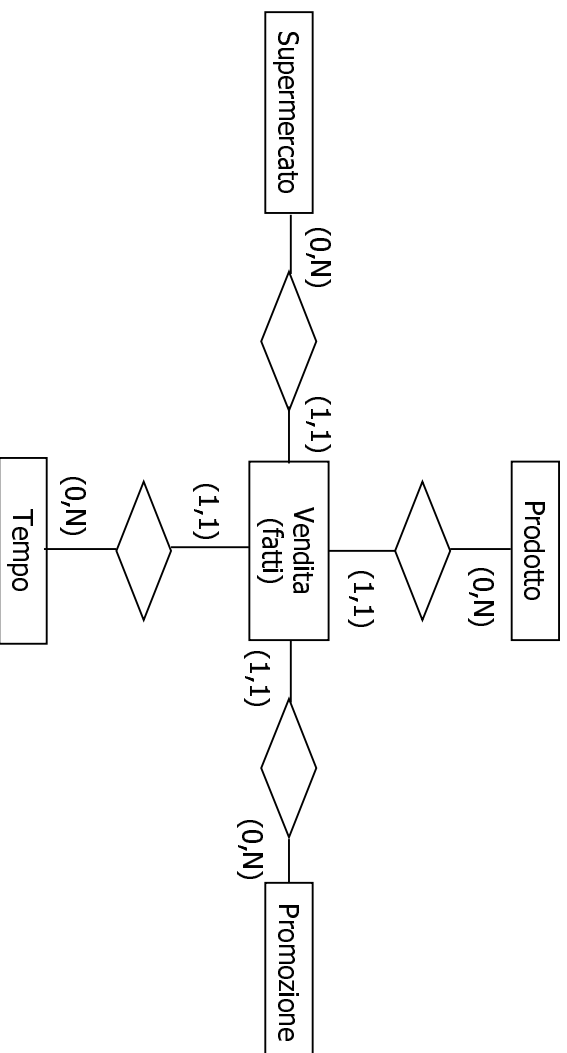
- Corrisponde ad un diagramma ER con struttura semplice
 - Una entità centrale rappresenta i **fatti**
 - varie entità disposte a raggiera rispetto all'entità centrale rappresentano le **dimensioni** dell'analisi (≥ 2)
 - relazioni 1:N collegano ogni istanza di fatto ad una sola istanza di ciascuna delle dimensioni



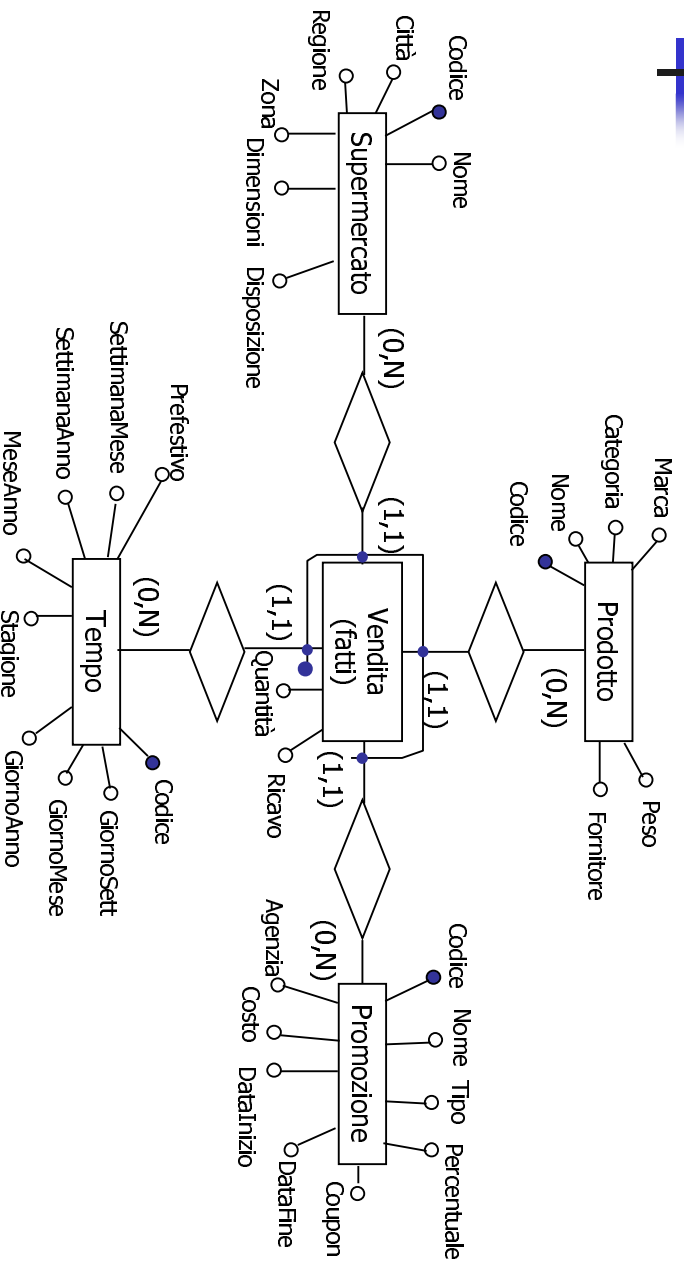
Analisi dei rimborsi (stella)



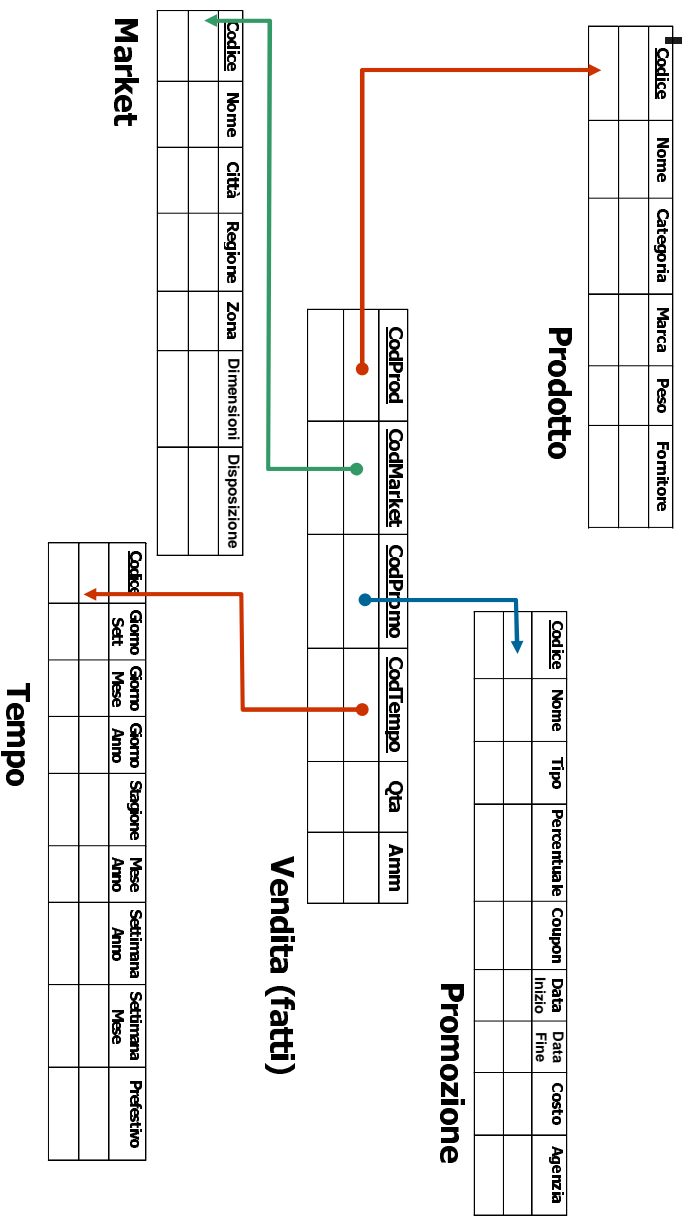
Analisi delle vendite (stella)



Analisi delle vendite (ER)



Analisi delle vendite (Logico)



Ridondanze...

- Le dimensioni presentano spesso ridondanze e dati derivati
 - La ridondanza è introdotta volutamente per rendere più efficienti le operazioni di analisi dei dati

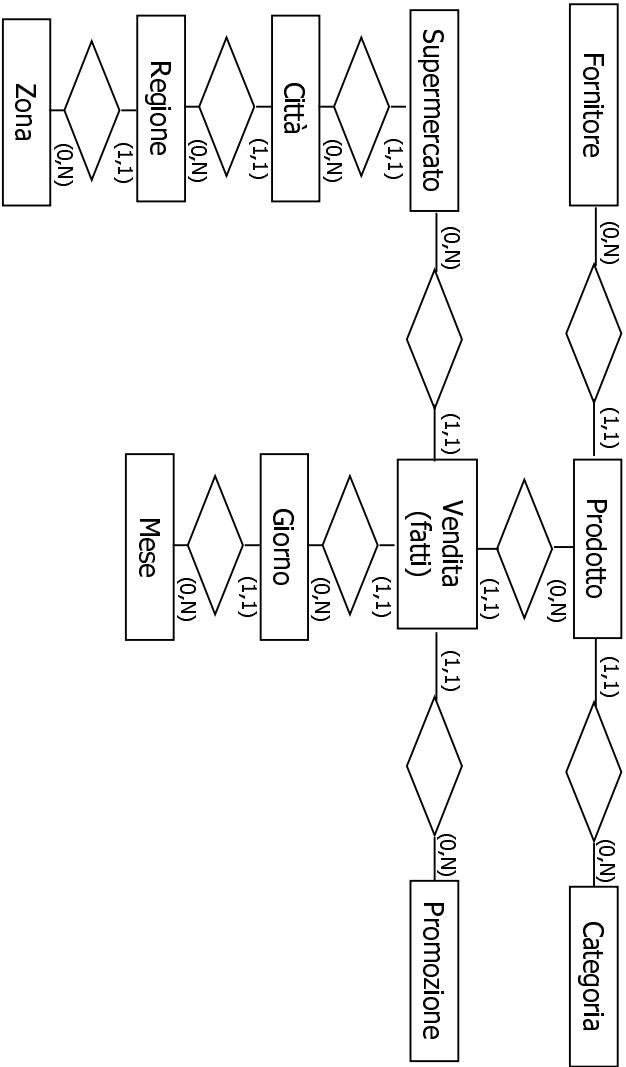
Tempo									
Codice	Giorno	Giorno	Giorno	Stagione	Mese	Settimana	Settimana	Prefestivo	
	Sett	Mese	Anno		Anno	Anno	Mese		

Basterebbe la data, ma per fare l'analisi si dovrebbe calcolare dalla data l'informazione di interesse (es. il giorno della settimana) per ogni tupla

- Quindi in generale le relazioni che rappresentano le dimensioni non sono normalizzate (presentano dipendenze funzionali che non coinvolgono la chiave). Es:

Giorno anno → Stagione

Diagramma a “fiocco di neve”



Fatti, misure e dimensioni

- La tabella dei fatti contiene degli attributi numerici (misure) che sono l’oggetto dell’analisi
- Le applicazioni di analisi aggregano i fatti in base a **criteri sulle dimensioni**

CodProd	CodMarket	CodPromo	CodTempo	Qta	Ammm

- Le tabelle delle dimensioni contengono invece prevalentemente informazioni testuali descrittive
- Gli attributi delle dimensioni sono usati per vincolare le richieste (fare un’analisi sulle dimensioni)

Analisi sulle dimensioni

Codice	Nome	Categoria	Marca	Peso	Fornitore

Prodotto

CodProd	CodMarket	CodPromo	CodTempo	Qta	Annm

Vendita (fatti)

Codice	Nome	Città	Regione	Zona	Dimensioni	Disposizione

Market

Aggregazione (somma) della quantità rispetto a Regione di vendita e Categoria di prodotto

Regione	Categoria	Qtot

```
select Regione,Categoria,sum(Qta)
from Vendita,Prodotto,Market
where (CodMarket=Market.Codice)
and (CodProd=Prodotto.Codice)
group by Regione,Categoria
```

Roll up

Codice	Nome	Categoria	Marca	Peso	Fornitore

Prodotto

CodProd	CodMarket	CodPromo	CodTempo	Qta	Annm

Vendita (fatti)

Codice	Nome	Città	Regione	Zona	Dimensioni	Disposizione

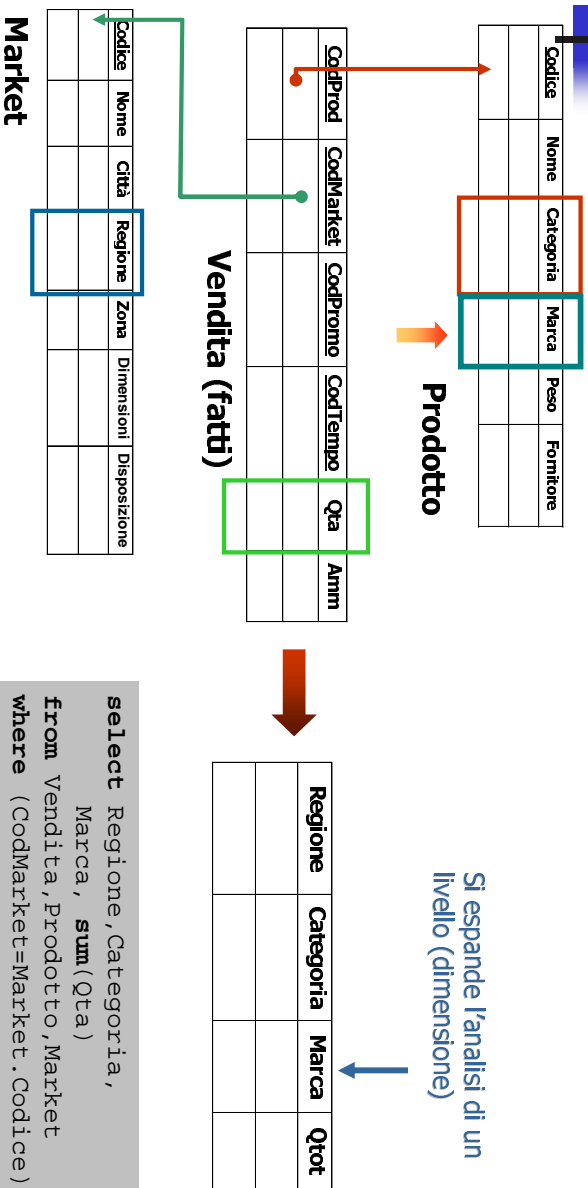
Market

Si elimina una dimensione di analisi

Regione	Categoria	Qtot

```
select Categoria,sum(Qta)
from Vendita,Prodotto
where (CodProd=Prodotto.Codice)
group by Categoria
```

Drill down



```
select Regione, Categoria,
       Marca, sum(Qta)
from Vendita, Prodotto, Market
where (CodMarket=Market.Codice)
and (CodProd=Prodotto.Codice)
group by Regione, Categoria, Marca
```

Viste materializzate

OLAP

- richiedono interrogazioni molto costose
- servono tecniche specializzate

Viste materializzate

- sono tabelle che contengono i dati corrispondenti ad una vista:
in questo caso un **data cube**
- sono utili perché permettono di **non ripetere** la stessa interrogazione
- l'aggiornamento dei dati comporta la modifica delle relative viste materializzate
 - particolarmente adatte ai casi in cui i dati subiscono poche modifiche
 - possono essere mantenute usando dei trigger



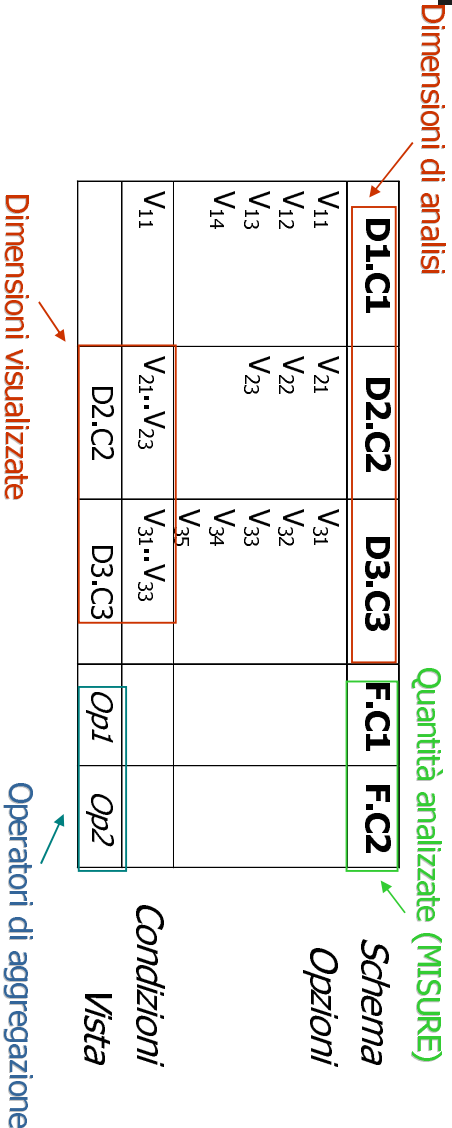
ROLAP vs MOLAP

Realizzazione di DataWarehouse

- **ROLAP (Relational OLAP)**
soluzione basata su un DBMS relazionale esteso (dati in tabelle ma organizzazione efficiente orientata all'analisi)
- **MOLAP (Multidimensional OLAP)**
Dati memorizzati in forma multidimensionale (prodotti specializzati)



Un'interfaccia utente



- I valori V_{ij} sono i possibili valori assunti dall'attributo Di
- Come condizione si possono definire intervalli di valori o uno specifico valore
- Quando si seleziona uno specifico valore non è significativo riportare la dimensione nella vista

In SQL

D1.C1	D2.C2	D3.C3	F.C1	F.C2
V ₁₁	V ₂₁	V ₃₁		
V ₁₂	V ₂₂	V ₃₂		
V ₁₃	V ₂₃	V ₃₃		
V ₁₄		V ₃₄		
		V ₃₅		
V ₁₁	V ₂₁ ..V ₂₃	V ₃₁ ..V ₃₃		
	D2.C2	D3.C3	Op1	Op2

Schema
Opzioni

Condizioni
Vista

```
select D2.C2, D3.C3, Op1(F.C1), Op2(F.C2)
from Fatti as F, Dimensione1 as D1, Dimensione2 as D2,
Dimensione3 as D3
where predicato-join(F,D1) and predicato-join(F,D2) and
predicato-join(F,D3) and condizioni-selezione-su-valori
group by D1.C1,D2.C2,D3.C3
order by D1.C1,D2.C2,D3.C3
```

Output

- I risultati dell'interrogazione si possono visualizzare
 - Relazione

D2.C2	D3.C3	F.C1	F.C2
V ₂₁	V ₃₁	R ₁₁	R ₂₁
V ₂₃	V ₃₁	R ₁₂	R ₂₂
V ₂₃	V ₃₂	R ₁₃	R ₂₃
V ₂₃	V ₃₃	R ₁₄	R ₂₄

- Tabella di foglio elettronico

	V ₃₁	V ₃₂	V ₃₃
V ₂₁	R ₁₁	-	-
V ₂₂	-	-	-
V ₂₃	R ₁₂	R ₁₃	R ₁₄

F.C1

	V ₃₁	V ₃₂	V ₃₃
V ₂₁	R ₂₁	-	-
V ₂₂	-	-	-
V ₂₃	R ₂₂	R ₂₃	R ₂₄

F.C2



Data mining



Data Mining

Il processo di Data Mining

- consente di **estrarre automaticamente informazione** da un insieme di dati
- l'informazione è **nascosta** a causa di
 - la quantità di dati: ad. es. transazioni delle carte di credito, delle compagnie telefoniche, ...
 - la loro complessità: ad es. occorre integrare sorgenti di informazioni diverse fra loro, non ci sono noti i fattori che influenzano quello che si cerca,
 - la velocità a cui arrivano: ad es. per le carte di credito possono essere decine di transazioni al secondo,...
- E' l'ultimo stadio del processo di analisi (si usa a valle degli OLAP)
- Può fornire un importante ritorno economico



Applicazioni

Vendita al dettaglio e marketing

- Scoperta delle abitudini dei clienti
- Scoperta delle associazioni fra le caratteristiche demografiche dei clienti
- Predizione della risposta alle campagne pubblicitarie
- Analisi delle associazioni fra i prodotti acquistati (market basket)

Banche

- Uso fraudolento delle carte di credito
- Individuare i clienti che stanno per cambiare carta di credito, i clienti fedeli,...
- Determinare la quantità d'uso della carte di credito per gruppi di clienti



Applicazioni II

Assicurazioni

- Analisi delle richieste di risarcimento
- Predire quali clienti possono essere interessati a nuove tipologie di polizze
- Predire il rischio associato ad una polizza con nuovo cliente

Medicina

- Predire il rischio di una malattia associato ad ogni paziente
- Predire la migliore cura per un determinato paziente



Applicazioni III

Bioinformatica

- Predire la cancerogenità di una molecola
- Predire l'efficacia di una molecola nella cura di una certa malattia
- Scoprire gruppi di molecole simili per le quali ci si aspetta proprietà simili

Applicazioni web

- In un servizio dedicato al cinema (libri, giochi, ..) , suggerire agli utenti nuovi film da vedere (libri da acquistare, giochi da provare,...)
- Individuare nel web le comunità che sono interessate allo stesso argomento
- In un forum di discussione individuare gli eventi, cioè i momenti in cui cambia drasticamente l'argomento di cui si discute



Il processo di knowledge discovery e quello di data mining

Il processo di knowledge discovery è suddiviso nelle seguenti fasi

- **Selezione dei dati**
Si scelgono i dati da analizzare. Essi possono provenire da un OLTP o da un OLAP
- **Ripulitura dei dati e trasformazione**
Occorre ripulire i dati e prepararli per le operazioni successive. Spesso le tabelle sono denormalizzate e combinate in un'unica tabella
- **Data mining**
Si applicano tecniche di apprendimento automatico, clustering,
- **Valutazione e interpretazione**
Nella maggior parte dei casi i risultati prodotti dal data mining non sono abbastanza affidabili da essere usati direttamente. Essi devono essere valutati e interpretati.



Tecnologie per il data mining

- Si usano tecniche provenienti dall'intelligenza artificiale
 - tali tecniche sono adattate per migliorarne le prestazioni su grandi quantità di dati
- Esistono numerosi tool per il data mining, ma
 - ogni applicazione ha una soluzione differente
 - per trovare una buona soluzione occorrono degli "artigiani" che selezionino la strada giusta fra un ampio insieme di tecnologie
- Le tecnologie per il data mining
 - permettono di scoprire informazione che in altri modi non è accessibile: sapere qualcosa che nessuno sa può essere un vantaggio enorme
 - sono molto costose da implementare



Tipologie di applicazioni

Analisi delle associazioni

- individuare le regole nascoste del tipo: l'evento A implica l'evento B
 - ad es. chi compra una stampante di solito compra anche il toner

Problemi di classificazione o regressione

- a partire da un insieme di esempi si apprende a classificare un oggetto
 - ad es. si vuol classificare un nuovo utente di un'assicurazione come utente ad alto rischio o meno: addestra un modello con gli esempi dei vecchi clienti

Problemi di clustering

- Si cerca di organizzare automaticamente gli eventi/oggetti di un database
 - ad es. si vuol identificare le molecole con un proprietà farmacologiche simili

Scoperta degli eventi che deviano dal comportamento normale

- Si cerca di individuare gli eventi, gli oggetti i comportamenti anomali
 - ad es. si vuol individuare le frodi su una carta di credito

Analisi delle associazioni: il problema del carrello

Il problema del carrello del supermercato

- Data la registrazione delle “transazioni” di un supermercato:
 - una transazione è un insieme di oggetti acquistati contemporaneamente da un utente
- trovare gli oggetti che più di frequente sono stati acquistati insieme
 - ad es. farina e lievito oppure farina, lievito, latte

TID	CID	Data	Prod.	Q.t à
111	201	5/1/05	farina	2
111	201	5/1/05	lievito	1
111	201	5/1/05	latte	3
111	201	5/1/05	carne	6
112	105	7/1/05	farina	1
112	105	7/1/05	lievito	1
112	105	7/1/05	latte	2
113	106	7/1/05	farina	2
113	106	7/1/05	latte	1
114	201	8/1/05	farina	3
114	201	8/1/05	lievito	2
114	201	8/1/05	carne	6
114	201	8/1/05	vino	6

Analisi delle associazioni: il problema del carrello II

Si definisce una misura

- Il supporto di un insieme di oggetti S: la percentuale delle transazioni in cui S è presente
 - ad es.
supporto({farina, lievito})=75%,
supporto({farina, lievito,latte})=50%

Usare l'SQL può non essere efficiente

```
select T1.prod, T2.prod, count(*) as N
from Transazioni as T1, Transazioni as T2
where T1.TID=T2.TID and not (T1.prod=T2.prod)
GROUP BY T1.prod, T2.prod
HAVING N >= 3
```

TID	CID	Data	Prod.	Q.tà
111	201	5/1/05	farina	2
111	201	5/1/05	lievito	1
111	201	5/1/05	latte	3
111	201	5/1/05	carne	6
112	105	7/1/05	farina	1
112	105	7/1/05	lievito	1
112	105	7/1/05	latte	2
113	106	7/1/05	farina	2
113	106	7/1/05	latte	1
114	201	8/1/05	farina	3
114	201	8/1/05	lievito	2
114	201	8/1/05	carne	6
114	201	8/1/05	vino	6

Trova gli insiemi di due
oggetti con supporto
maggiore del 75%



Analisi delle associazioni: il problema del carrello III

Soluzioni inefficienti per il problema del carrello (con molti dati)

- Si implementa e si ottimizza un'interrogazione SQL come una qualsiasi interrogazione
 - Inefficiente perché richiede l'implementazione di join e raggruppamento: due join per gruppi di due oggetti, tre join per gruppi di tre oggetti
- Si scorrono le transazioni in maniera sequenziale, tenendo dei contatori che contano tutti i gruppi di oggetti incontrati
 - inefficiente, perché i possibili gruppi possono essere numerosissimi: la struttura dati per memorizzare i contatori potrebbe essere più grande della memoria principale



Un algoritmo efficiente: Apriori

- Si trovano tutti gli insiemi più frequenti con n elementi utilizzando quelli con n-1 elementi
 1. Al primo passo si scandisce la tabella delle transizioni e si contano i singoli oggetti più comprati: si sceglie l'insieme I_1 di oggetti che contengono quelli il cui supporto è maggiore di una certa soglia T
 2. Si scandisce la tabella della transizioni e contano le occorrenze delle coppie che contengono oggetti di I_1 ; si sceglie l'insieme I_2 di coppie che contengono quelle il cui supporto è maggiore di T
 3. Per trovare gli insiemi di n oggetti si ripete 2 (si ottiene I_n combinando gli oggetti di I_{n-1})
- Al passo 2 si considera solo un sottoinsieme delle coppie (gruppi): se la soglia è alta, la struttura dati sta sicuramente in memoria
- Nota che ... se un gruppo di oggetti ha supporto maggiore di T anche tutti i sottogruppi hanno supporto maggiore di T
- Per gruppi di n oggetti, occorre scandire la tabella delle transizioni n volte!

Un algoritmo efficiente: Apriori

Ricerca di oggetti con supporto maggiore del 75%

```

Leggi le transizioni e calcola supporto(oj) per ogni oggetto o
for each oggetto o begin
    if supporto(oj) >= 75% then inserisci oj in Ij;
end

k=1;
repeat
    Leggi le transizioni e calcola supporto(C ∪ D) per ogni C ∈ Ij and D ∈ Ik
    for each C ∈ Ij and D ∈ Ik begin
        if supporto(C ∪ D) >= 75% then inserisci C ∪ D in Ik+1;
    end
end

```

Regole di associazione

Ricerca delle regole di associazione

- Consiste nell'identificare le regole di implicazione fra gli eventi $H \Rightarrow T$
 - Ad es., $\{farina\} \Rightarrow \{lievito\}$
- Per ogni regola $H \Rightarrow T$ si definiscono
 - $supporto(H \Rightarrow T) = supporto(H \cup T)$
Ad. es. $supporto(\{farina\} \Rightarrow \{lievito\}) = 75\%$
 - $confidenza(H \Rightarrow T) = supporto(H \Rightarrow T) / supporto(H)$
Ad. es. $confidenza(\{farina\} \Rightarrow \{lievito\}) = 0.75$

TID	CID	Data	Prod.	Q.t à
111	201	5/1/05	farina	2
111	201	5/1/05	lievito	1
111	201	5/1/05	latte	3
111	201	5/1/05	carne	6
112	105	7/1/05	farina	1
112	105	7/1/05	lievito	1
112	105	7/1/05	latte	2
113	106	7/1/05	farina	2
113	106	7/1/05	latte	1
114	201	8/1/05	farina	3
114	201	8/1/05	lievito	2
114	201	8/1/05	carne	6
114	201	8/1/05	vino	6

Regole di associazione II

Trovare le regole di associazione

- Si tratta di trovare tutte le regole R per le quali $supporto(R) > min_sup$ e $confidenza(R) > min_con$
- Si risolve basandosi sull'algoritmo per la ricerca degli insiemi frequenti

```
Trova tutti gli insiemi frequenti  $S$  per i quali  $supporto(S) > min\_sup$   
for each  $S$  begin  
  Dividi  $S$  in tutte possibili coppie di insiemi  $A, B, S = A \cup B$   
  for each  $A, B$  begin  
    if  $confidenza(A \Rightarrow B) > min\_con$   
      then  $A \Rightarrow B$  è una delle regole trovate  
    end  
  end
```

Classificazione (regression)

In cosa consiste

- consiste nell'inferire una proprietà di un oggetto sulla base di alcune sue caratteristiche
 - ad es. si vuol inferire il rischio di un utente di una polizza
- la proprietà da inferire può essere un valore numerico qualsiasi (regression) o appartenere ad un insieme finito (classificazione)

Nel nostro caso

- Spesso si crea una tabella che contiene tutte le proprietà necessarie all'inferenza
- La proprietà da inferire è un attributo della tabella

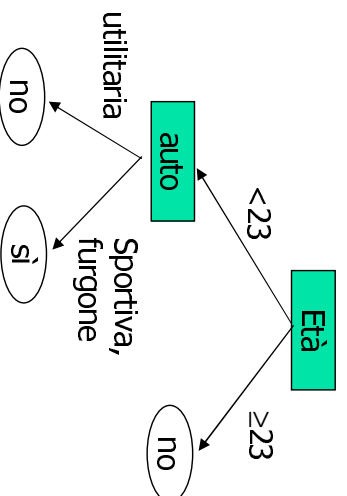
POLIZZE(id, nome,età,auto_o_furgone,cavalli,attività,.....,alto**Rischio**)

Caratteristiche

Proprietà da predire

Alberi di decisione

- Rappresentano un **insieme di regole** che permettono di fare la predizione automaticamente
 - Ogni nodo interno rappresenta un test e i suoi rami indicano le risposte
 - Ogni foglia rappresenta una decisione
- Sono costruiti automaticamente usando i dati disponibili
 - ad le caratteristiche di rischio dei vecchi clienti dell'assicurazione



Alberi di decisione II

La realizzazione dell'albero si basa su due fasi

- Costruzione
- Raffinamento

Costruzione

- Si cerca un buon criterio C per dividere il dataset in due sottoinsiemi D1, D2
 - Esistono criteri diversi per la divisione in sottoinsiemi: ad esempio si può scegliere l'attributo che massimizza l'information gain
 - un buon criterio dovrebbe minimizzare la profondità dell'albero

- Si costruisce un nodo che usa il criterio C e si applica ricorsivamente l'algoritmo a D1 e D2

Raffinamento

- L'albero costruito viene semplificato eliminando i rami meno importanti



Alberi di decisione III

```
buildTree(Nodo n, Partizione P, criterio di scelta S )
1.  Applica S a P per trovare il criterio di divisione
2.  If (esiste un buon criterio di divisione){
3.      crea due figli n1, n2
4.      dividi P in P1, P2
5.      buildTree(n1,P1,S)
6.      buildTree(n2,P2,S)
}
```



Scelta del criterio

- Per scegliere un criterio, occorre valutarlo su una tabella che contiene i dati: cosa succede se la tabella è troppo grande e non entra in memoria principale ?
- Si costruisce l'**AVC set** (Attribute Value Class label): contiene le occorrenze di ogni coppia attributo-classe
- L'AVC set è tipicamente molto più piccolo della tabella
- L'AVC set viene usato per calcolare il criterio
 - Si osservi che questo è possibile perché il criterio usa un solo attributo e un solo valore

AVC set: un esempio

POLIZZE

età	auto	alto_rischio
20	furgone	sì
30	sportiva	sì
20	utilitaria	no
20	utilitaria	sì
40	furgone	sì
20	sportiva	sì

```
SELECT età, alto_rischio , count(*)  
FROM polizze  
GROUP by età, alto_rischio  
  
SELECT auto, alto_rischio , count(*)  
FROM polizze  
GROUP by auto, alto_rischio
```

AVC set

età	alto_rischio	occorrenze
20	sì	3
20	no	1
30	sì	1
40	sì	1

auto	alto_rischio	occorrenze
furgone	sì	2
utilitaria	sì	1
utilitaria	no	1
sportiva	sì	2

Reti neurali

Cosa sono

- sono modelli parametrici in grado di implementare una funzione $f_w(x_1, x_2, \dots, x_n)$
 - w sono i parametri da apprendere
 - x_1, x_2, \dots, x_n le caratteristiche note (ad es. età, auto posseduta, ... di un cliente)
 - $f_w(x_1, x_2, \dots, x_n)$ calcola la proprietà da predire
- i parametri sono appresi da esempi

Osservazione

- Poichè l'apprendimento può richiedere molto tempo con le reti neurali, non si usano tutti i dati disponibili, ma sono un sottoinsieme selezionato un modo casuale

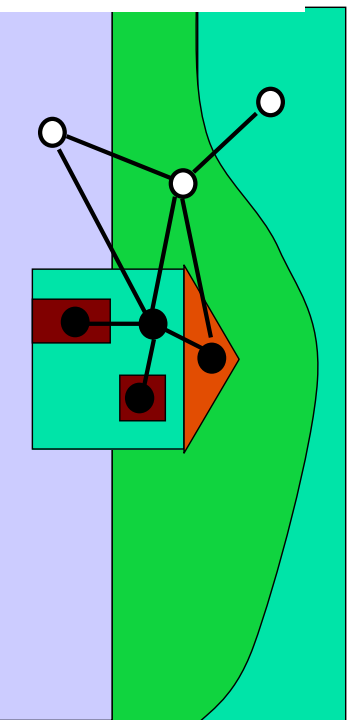
Reti neurali per l'elaborazione di grafi: Graph Neural Networks

- Sono un'evoluzione delle reti neurali che possono essere applicate su insieme di grafi
 - Il dataset rappresenta un insieme di oggetti (nodi) correlati da relazioni (archi)
 - Ogni nodo ha delle etichette che rappresentano caratteristiche dell'oggetto
 - si tratta di predire una proprietà degli oggetti usando sia le caratteristiche degli oggetti che le loro relazioni

Esempio

Localizzare una casa, distinguendo le regioni (nodi neri) che la compongono dalle altre regioni (nodi bianchi)

L'apprendimento usa immagini di case e non



Relational learning

- Il relational learning consiste nel prevedere un attributo di una relazione utilizzando gli altri attributi e i collegamenti con le altre relazioni
- Il relational learning può essere affrontato con tecniche di programmazione logica e con le graph neural networks

Esempio: prevedere il livello di rischio di una polizza

Assicurati			Polizze			Rimborsi	
nome	età		tipo	oggetto assicurato	livello rischio	motivo	q.tà
paperino	35		RCA	fiat punto	?	incidente	100
pippo	40		vita	Pippo	?	atti vand.	300
			RCA	cinquecento	?		

Clustering

In cosa consiste

- mira a suddividere un insieme di oggetti in modo che
 - oggetti nello stesso gruppo siano simile
 - oggetti in gruppi diversi siano dissimili
- Il raggruppamento viene attuato con tecniche di apprendimento **non supervisionato**

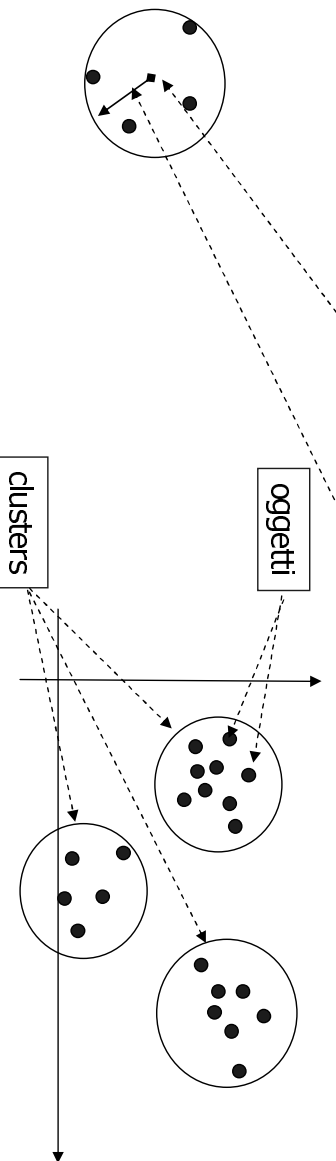
Applicazioni

- Individuazione di molecole con proprietà curative simili
- Raggruppamento di utenti in base al loro comportamento su un sito
- Raggruppamento di utenti in base alle loro caratteristiche sociali ed economiche

Clustering II

Gli algoritmi tipici di clustering

- gli oggetti da organizzare sono punti in uno spazio n-dimensionale
- esiste una misura che definisce la distanza fra gli oggetti
- l'algoritmo deve individuare delle sfere **che racchiudano gli oggetti**
- ogni cluster ha un **centro** e un **raggio**



Un algoritmo efficiente: BIRCH

- Occorrono algoritmi di clustering specifici per trattare grandi moli di dati
- Gli algoritmi classici (ad esempio, k-means) non sono adatti perché prevedono un alto numero di epoche di apprendimento: ogni epoca implica una lettura del training set
- L'algoritmo **BIRCH** legge una sola volta tutti gli oggetti e produce k clusters $(c_1, r_1), (c_2, r_2), \dots, (c_k, r_k)$
 - c_1, c_2, \dots, c_k sono i centri dei clusters, r_1, r_2, \dots, r_k i raggi
 - k è scelto in modo tale che la definizione dei clusters possa essere tenuta in memoria
 - esiste un parametro ϵ che definisce la massima dimensione di un cluster

Un algoritmo efficiente: BIRCH

```
repeat
    leggi il record corrente  $A$  e trova il cluster  $i$  più vicino ad  $A$ 
    prova ad inserirvi  $A$  e calcola il nuovo centro  $nc_i$  e la distanza  $d$  di  $A$  a  $nc_i$ 
    if  $d < \epsilon$  then inserisci  $A$  nell' $i$ -esimo cluster
        else crea un nuovo cluster con centro  $A$ 
    end
until  $i$  si sono letti tutti i record
```

if si è raggiunto il massimo numero di clusters
 then incrementa ϵ ed, eventualmente, fonda i clusters;
 vai prossimo record



Strumenti per il data mining

Strumenti costruiti appositamente

- alcuni produttori costruiscono strumenti ad hoc per il data mining, capaci di prendere dati da sorgenti diverse
 - ad. es. SAS Enterprise Miner, SPSS Clementine, CART (Salford Systems), Megaputer PolyAnalyst, ANGOSS KnowledgeStudio

Strumenti associati ai DBMS

- i maggiori produttori di DBMS offrono anche strumenti per il data mining
 - IBM Intelligent Miner
Supporta numerosi algoritmi per la ricerca di regole di associazione, la classificazione, la regressione e il clustering
 - Microsoft Analysis Service
Supporta gli alberi di decisione, il clustering



Database e Web

Applicazioni e DBMS

Un'applicazione (per l'accesso a database) contiene tre elementi

- un'interfaccia utente
- i dati
- una logica

Si può usare un'architettura

- a due strati (2-tier)
 - uno strato contiene la logica e l'interfaccia
 - uno strato contiene i dati
- a tre strati (3-tier) o più (n-tier)
 - uno strato diverso per ciascun elemento

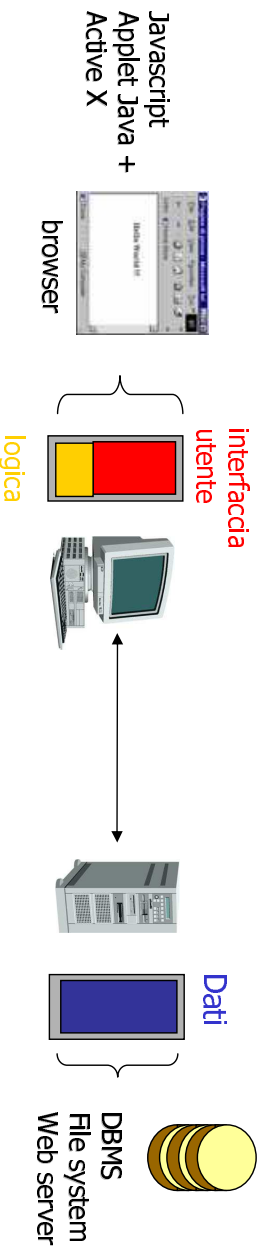
Architettura a due strati

Il client: interfaccia utente e logica di controllo

- Un'applicazione sviluppata con
 - un comune linguaggio di programmazione
 - browser + client side scripting (Javascript, Applet Java, Active X)

Il server: i dati

- un DBMS e il file system
- eventualmente un Web server (se è un'applicazione Web)



Architettura a due strati II

Il client accede ai dati attraverso

- API proprietarie del DBMS
- connettori standard (JDBC, ODBC, ...)

Osservazioni

- puo' anche accadere che parte della logica sia realizzata sul lato server
- l'architettura 2-tier
 - è semplice da realizzare
 - difficile da espandere
 - tipica delle applicazioni tradizionali

Architettura a tre strati

Presentation tier

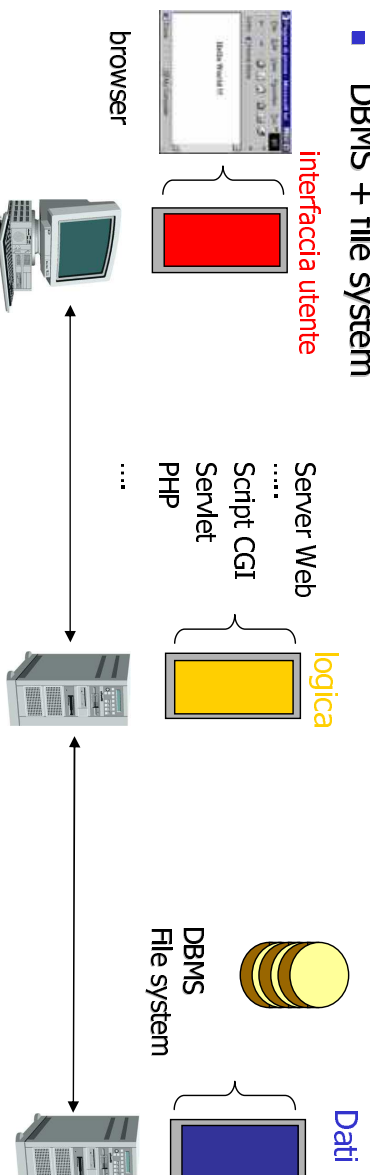
- un browser

Business logic tier (middle tier)

- un server Web, un transaction server, un application server....
- server side logic (CGI, PHP, servlet, JSP,..)

Data tier

- DBMS + file system





Architettura a tre strati II

Osservazioni

- si possono avere architetture **n-tier**
- non è sempre semplice distinguere cosa deve essere messo su un certo livello

Vantaggi

- **separazione** logica di controllo da problematiche di presentazione
 - Si può cambiare la logica di controllo senza cambiare l'interfaccia
 - Si possono riusare gli stessi oggetti di controllo per applicazioni diverse
- la gestione dell'applicazione è **centralizzata**



Architettura a tre strati III

Vantaggi

- **thin client** (client leggeri)
 - semplificata amministrazione dei client (.. solo un browser)
 - bassi costi dell'hardware per il client
- carico di lavoro facile da bilanciare

Svantaggi

- architettura più complessa
- si usano molti strumenti per realizzare la logica di controllo



Web Information Systems

Web Information Systems (WIS)

- sistemi informativi accessibili attraverso un **browser**
- permettono un accesso “universale” all'informazione
- nati per il Web, sono usati anche
 - in **Intranet**
 - per realizzare applicazioni di ogni tipo
- strettamente legati ai DBMS
 - permettono di accedere l'enorme quantità di informazione preesistente nei database
 - man mano che diventano più complessi, parte dell'informazione originariamente memorizzata in pagine HTML viene spostata in database



Accesso a database tramite Web

Vantaggi

- indipendenza dalla piattaforma
 - dei client e parzialmente dei server
 - supporto per l'accesso contemporaneo da piattaforme diverse
 - visibilità 'world wide
- interfaccia grafica
 - facile da realizzare
 - standard
- accesso alla rete trasparente
- applicazioni facilmente aggiornabili e scalabili

Accesso a database tramite Web II

Svantaggi

- basso livello di
 - affidabilità'
 - sicurezza
- alto costo di sviluppo di un'applicazione
- l'HTML e HTTP sono **poveri di funzionalità'**
 - es. connessioni statless, difficoltà' nell'interagire con l'utente, ...
- **Lentezza**
 - Internet è lenta
 - Gli interpreti dei linguaggi usati sono lenti
- scalare un sito non è banale, se un solo server Web non è sufficiente

Funzionamento di un server Web

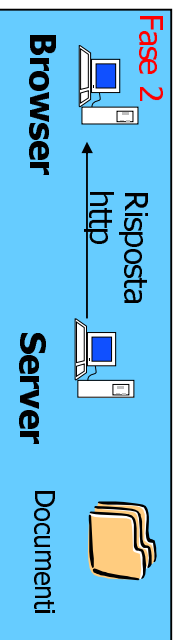
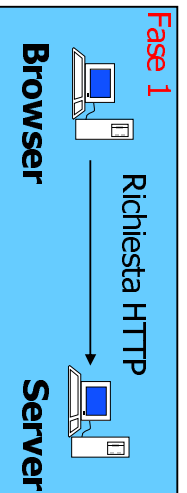
Quando si clicca sul link "http://www.ing.unisi.it/eventi.html"

Prima fase: il browser

- prepara una richiesta secondo il protocollo HTTP
- apre una connessione con il server `www.ing.unisi.it` ed invia la richiesta per avere il documento `/eventi.html`

Seconda fase: il server

- prepara una risposta nel formato HTTP che contiene
 - il documento HTML in `/eventi.html` se esiste
 - oppure contiene un messaggio di errore
- invia la risposta e chiude la **connessione**





Funzionamento di un server Web II

Terza fase: il browser

- legge la risposta estraendo il documento inviato dal server
- mostra a video il documento secondo regole che dipendono dallo specifico browser

Osservazioni

- il server Web è solo un “passa carte”:
 - non fa altro che inviare i documenti richiesti
 - non conosce il contenuto dei documenti
- le connessioni sono stateless
 - per mantenere memoria dello stato occorre usare dei cookie
- I documenti HTML sono statici
 - occorre estendere HTML e/o i server Web se si vuole connettersi ad un DBMS



Estendere HTML e i server Web

Estensioni dei server Web

- Common Gateway Interface (CGI), Java Servlets
 - i documenti (HTML o altro) sono prodotti da (l'output) di programmi
- Estensioni proprietarie dei Web server (Netscape API, IIS API)
 - CGI e Servlets possono accedere a funzionalità messe direttamente a disposizione dal server Web
- linguaggi di scripting lato server (PHP, ASP, JSP)
 - si inserisce all'interno di HTML degli script che verranno eseguiti dal server Web (o da un'applicazione server) producendo documenti HTML puri

Estensioni di HTML

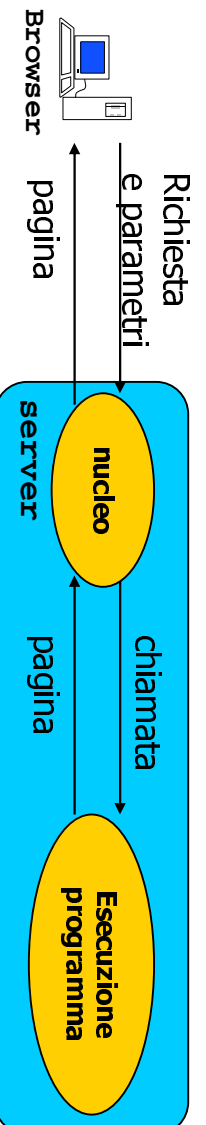
- linguaggi di scripting lato client (JavaScript, VBScript)
 - si inserisce all'interno di HTML degli script che verranno eseguiti dal browser
- Java applet
 - vere e proprie applicazioni che possono essere scaricate ed eseguite dal browser

Common Gateway Interface (CGI)

- è un protocollo
 - consente al Web Server di eseguire programmi esterni in grado di produrre pagine dinamicamente
 - definisce un insieme di **variabili di ambiente** utili alla applicazione (es., i parametri inviati dal client)
- l'applicazione
 - può essere scritta in un **qualsiasi linguaggio** (C, Java, linguaggi per script)
 - gli eseguibili devono essere inseriti in una directory gestita dal Web Administrator (/cgi-bin)

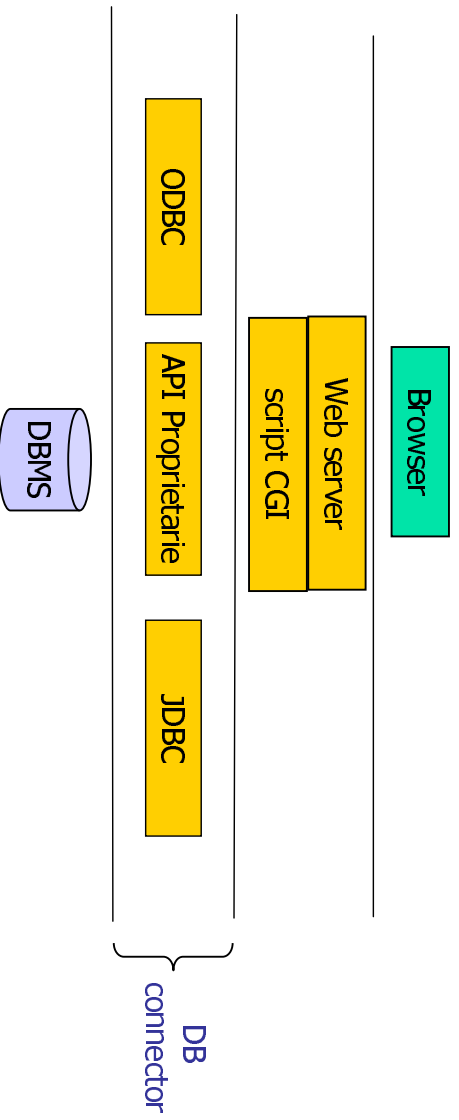
Common Gateway Interface II

- Il funzionamento
 - il browser invia al server una richiesta in cui si fa riferimento ad un cgi es. www.search.com/search.cgi?Picasso
 - il server chiama il programma "search.cgi"
 - il parametro "Picasso" è passato attraverso lo standard input (metodo POST nel form) o settando una variabile d'ambiente (metodo GET)
 - Il programma [search.cgi](#) produce in uscita un documento HTML e termina
 - Il server Web invia il documento al browser



Common Gateway Interface III

- connessione ad un DBMS
 - attraverso qualsiasi meccanismo disponibile nel linguaggio di programmazione prescelto
 - ODBC, JDBC, API proprietarie



DB connectors

API proprietarie

- moduli che forniscono le funzionalità necessarie a connettersi ed accedere ai dati di **uno specifico DBMS**
- realizzati dai produttori dei DBMS per i principali linguaggi

Connettori standard

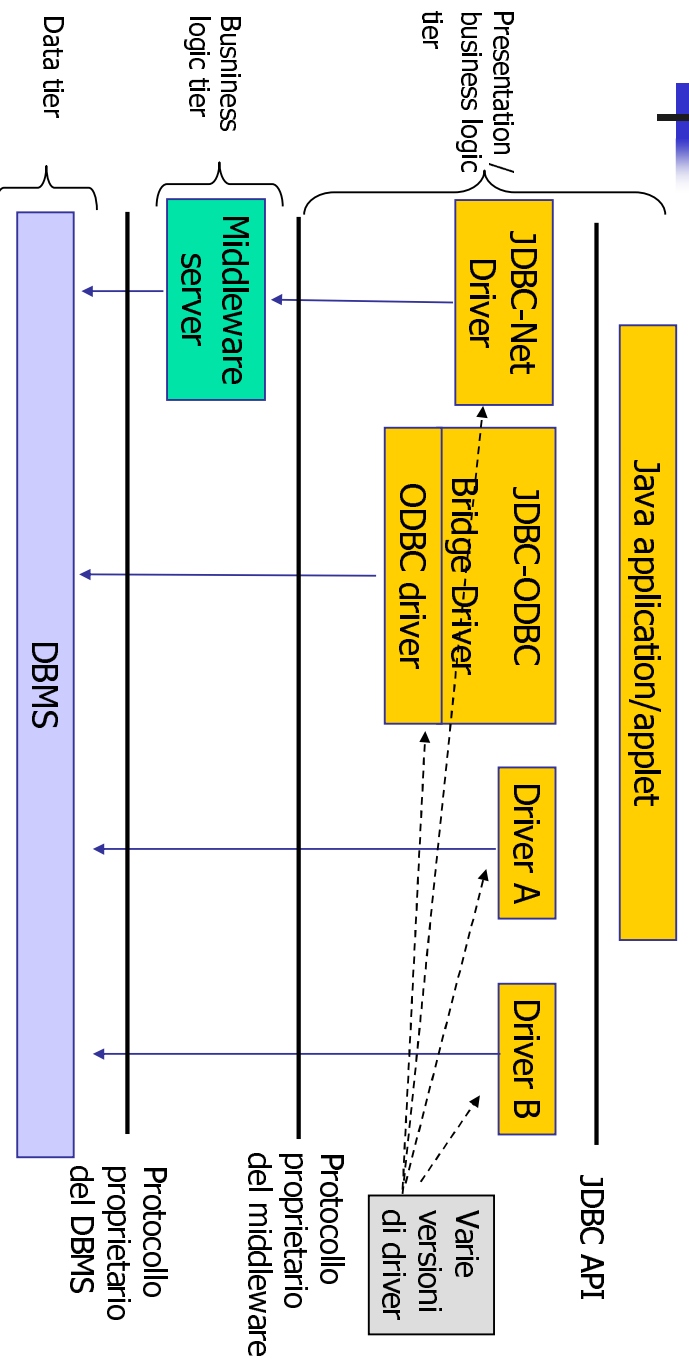
- moduli che forniscono le funzionalità per connettersi a **molti DBMS**
- **JDBC (Java DataBase Connectivity)**
 - Package di Java (usabile solo con Java)
 - disponibili driver su tutte le piattaforme per i principali DBMS SQL
- **ODBC (Microsoft Open Database Connectivity)**
 - scritto in C++
 - adatto per client Windows
 - disponibili driver per i principali DBMS SQL

Un esempio: JDBC

Quattro tipi di **driver JDBC**

- **Tipo 1: Bridge ODBC più driver ODBC**
Converte le chiamate ai metodi JDBC in chiamate a ODBC
- **Tipo 2: Driver parzialmente in Java più API native**
Converte le chiamate ai metodi JDBC in chiamate alle librerie del DBMS
- **Tipo 3: Driver Java puro più middleware**
Comunica con un server intermedio attraverso un protocollo proprietario
- **Tipo 4: Driver Java puro con protocollo proprietario**
Comunica con un protocollo proprietario direttamente con il DBMS

Un esempio: JDBC II



CGI: un esempio

I CGI e form: un uso tipico

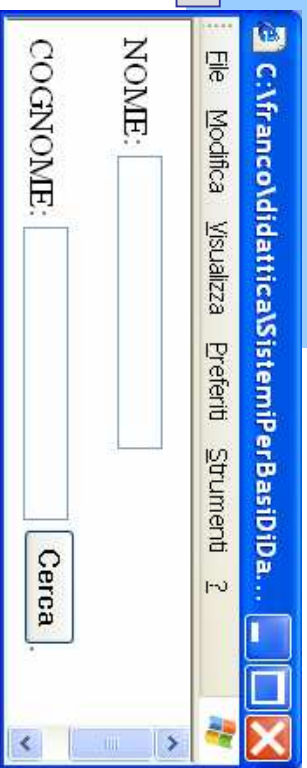
- il form raccoglie login e password dell'utente
- e chiama la procedura /cgi-bin/listaStudenti.cgi

```
<HTML>
<BODY>
<FORM ACTION="/cgi-bin/listaStudenti.cgi" METHOD="POST">
  NOME:<INPUT TYPE="text" NAME="nome" VALUE=""><P>
  COGNOME:<INPUT TYPE="text" NAME="cognome" VALUE="">
  <INPUT TYPE="submit" VALUE="Cerca">. <P>
</FORM>
</BODY>
</HTML>
```

i campi da leggere

il cgi da attivare

il metodo
di invio dei parametri



CGI: un esempio II

Un CGI in Java

- si crea una classe di nome listaStudenti
- i parametri in ingresso
 - se il FORM usa il metodo POST, vengono letti dallo standard input
 - se il FORM usa il metodo GET, vengono letti da una variabile d'ambiente

```
import java.cgi.lib.*;

class listaStudenti {

    public static void main( String args[] ) {

        Hashtable form_data = cgi_lib.ReadParse(System.in);

        String nome = (String)form_data.get("nome");
        String cognome = (String)form_data.get("cognome");
```

si leggono i parametri
dallo standard input

CGI: un esempio III

Per connettersi ad un DBMS occorre

- Caricare il driver JDBC che si intende usare

```
String driver= "COM.ibm.db2.jdbc.net.DB2Driver";  
Class.forName(driver).newInstance();
```

- Aprire una connessione con il DBMS

```
String url= "jdbc:db2://segreteria.ing.unisi.it/stud";  
String userid= "pippo";  
String passwd= "pippoPass";  
Connection con = DriverManager.getConnection(url, userid, passwd);
```

Driver, server
e database

CGI: un esempio IV

- Per eseguire un'interrogazione occorre creare uno **statement** ed eseguirlo

```
Statement stmt = con.createStatement();  
ResultSet rs = stmt.executeQuery("select cognome,nome,dataNascita"+  
    "from studenti"+  
    "where nome like %"+ nome+ "% and "+  
    "cognome like %"+ cognome+"%");
```

- **executeQuery** invia il comando al DBMS e pone il risultato in un oggetto di tipo **ResultSet**

CGI: un esempio V

- L'oggetto **ResultSet** consente di scorrere le righe e accedere agli attributi
- I tipi di dati SQL sono trasformati in tipi di dati Java

```
System.out.println("<HTML><BODY><OL>");  
while (rs.next()) {  
    String cdgnome = rs.getString("cognome");  
    int nome = rs.getString("nome");  
    Date dn = rs.getDate("dataNascita");  
    System.out.println("<LI>" + cognome + " " + nome + " "  
        + dn.toString());  
    System.out.println("</OL></BODY></HTML>");  
}
```

stampa inizio e
fine della pagina

stampa della
lista dei nomi

Iterazione su tutte le tuple
del risultato dell'interrogazione



Alternative a JDBC

Per Java esistono alternative a JDBC

- **SQLJ**
 - una versione di **embedded SQL** specializzato per JAVA
 - il codice SQL è scritto direttamente all'interno di Java
 - un precompilatore trasforma l'SQL in chiamate al DBMS
- **JAVA Blend**
 - crea una connessione diretta fra **oggetti JAVA** e **oggetti del database**: si opera sulle tabelle come se fossero oggetti
- **SQLJ** e **JAVA Blend** si appoggiano su JDBC



SQL: un esempio

I comandi SQL sono identificati da "#sql"

```
...
#sql iterator studentDetails(String nome, String cognome);
studentDetails studentIterator=null;
#sql studentIterator ={select cognome,nome from studenti};

System.out.println("<HTML><BODY><OL>");
while (studentIterator.next()) {
    System.out.println("<LI>"
        + studentIterator.nome()+ " "
        + studentIterator.cognome()+ " ");
}
System.out.println("</OL></BODY></HTML>");
rs.close();
```



SQL vs JDBC

- SQL facilita l'analisi statica
 - della sintassi
 - dei tipi
 - dello schema
- SQL permette la generazione di strategie di accesso da parte del DBMS in modo trasparente
- SQL richiede una fase precompilazione



Limiti dei CGI

CGI è molto semplice da usare ed è uno standard diffuso, ma

- ad ogni accesso allo script (programma) CGI,
 - si crea un nuovo processo
 - si apre e richiude una connessione con il DBMS
- tutte le comunicazioni fra client e CGI passano dal Web server
- ci sono problemi di sicurezza

Per ovviare a questi problemi

- i server Web forniscono nuove API: Netscape Server API, IIS API
 - gli script CGI sono caricati come parte del server Web e rimangono attivi fra una transazione e l'altra
 - gli script possono accedere a funzionalità per la connessione ai DBMS, per l'autenticazione e la gestione delle connessioni



Servlets

Cosa sono

- sono applicazioni Java in esecuzione su una JVM residente sul server
- realizzate attraverso il package Java: java.servlet

Come CGI

- il client invoca la servlet nel contesto di una FORM HTML
- la servlet esegue le operazioni richieste
- la servlet ridirige l'output al client in forma di pagina HTML

A differenza di CGI

- la servlet corrisponde ad un processo che viene caricato solo una volta e utilizzato per eseguire più operazioni distinte
- il server Web deve essere esteso con un servlet engine

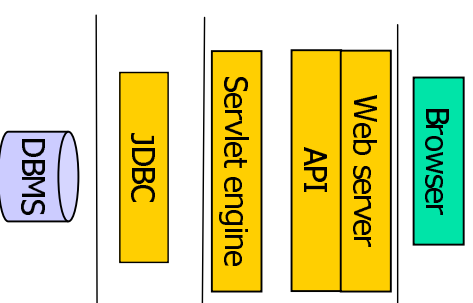
Servlets e DBMS

L'accesso a DBMS

- avviene attraverso JDBC, SQL, JavaBlend
- poiché' la servlet rimane attiva, le connessioni con il DBMS sono aperte una sola volta, solo al momento del caricamento

Osservazioni

- le servlet sono indipendenti dalla piattaforma
- permettono l'aggiunta di ulteriori strati all'architettura



Servlets: un esempio

Una servlet deve contenere i metodi

- **init()**
 - chiamato dal server nel momento del caricamento della servlet
- **destroy()**
 - chiamato dal server per distruggere la servlet: es. dopo un periodo di inattività
- **doPost()** e **doGet()**
 - chiamati tutte le volte che un browser richiede l'esecuzione della servlet

```
public class ListaStudenti extends HttpServlet {  
    init() {...}  
    destroy() {...}  
    doGet() {...}  
    doPost() {...}  
}
```

Servlets: un esempio II

- L'inizializzazione contiene la connessione al DBMS

```
String driver= "COM.ibm.db2.jdbc.net.DB2Driver";
String url= "jdbc:db2://segreteria.ing.unisi.it/stud";
:
:
:

public void init()throws ServletException {
try{

    class.forName(driver).newInstance();

    Connection con = DriverManager.getConnection(url, userid, passwd);

    PreparedStatement prep = con.prepareStatement(

        "select cognome,nome,dataNascita"+
        "from studenti"+
        "where nome like '%" +
        "cognome like '%" );

    catch (SQLException ex) {...}
}
```

Apertura
connessione

creazione
prepared
statement

Servlets: un esempio II

```
Public void doPost(HttpServletRequest req,HttpServletResponse res)
throws ServletException, IOException {

    res.setContentType("text/html");
    PrintWriter toClient = res.getWriter();
    String nome = req.getParameter("nome");
    String cognome = req.getParameter("cognome");

    toClient.println("<HTML><BODY><OL>");
    prep.setString(1,nome); prep.setString(2,cognome);
    resultSet rs=prep.executeQuery();

    while (rs.next()) {
        String c = rs.getString("cognome");
        String n = rs.getString("nome");
        Date d = rs.getDate("dataNascita");
        toClient.println("<LI> " + c + " " + n = " " + d.toString());}

    toClient.println("</OL></BODY></HTML>");
    toClient.close();}
```

preparazione
stream di uscita

Lettura parametri

esecuzione
interrogazione

chiusura connessione
con il browser

Linguaggi di scripting lato server

Cosa sono

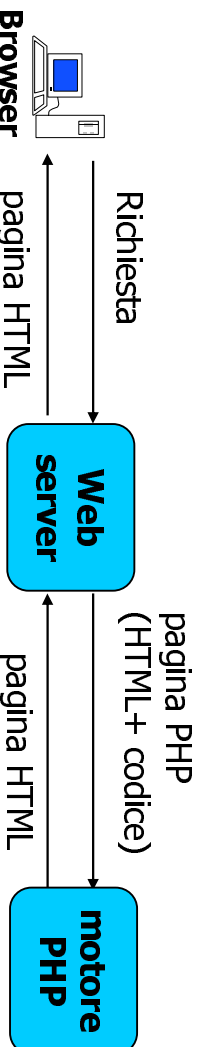
- linguaggi che permettono di definire **pagine dinamiche**
 - ASP, PHP, Active Perl,...
- Una pagina dinamica è costituita da
 - **HTML**
 - **codice** compreso tra tag speciali (<?php, ?> ..)

```
<HTML>
<TITLE>La pagina di Mago Mago' <TITLE>
<BODY>
  Ecco l'ambo da giocare:
  <?php
    srand((double)microtime()*1000000);
    $n1=rand(1,90);
    $n2=rand(1,90);
    while($n1=$n2){
      $n2=rand(1,90);
    }
    echo "$n1 $n2 <P>";
  ?>
</BODY>
</HTML>
```

Linguaggi di scripting lato server

Come funzionano

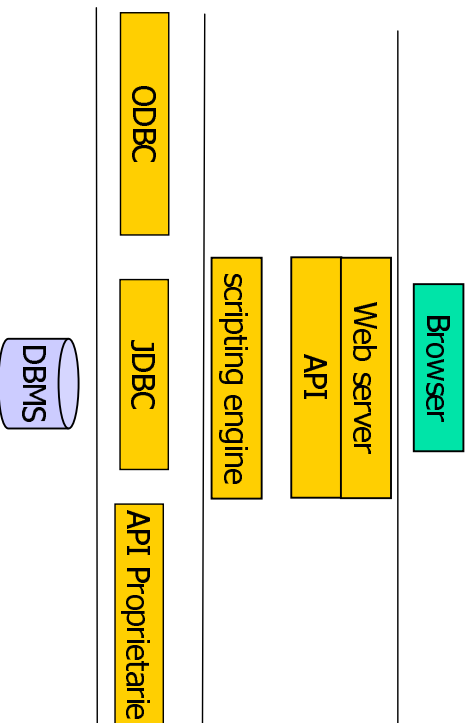
- Il Web Server è **esteso** con un **motore di scripting** (Script Engine)
 - quando arriva una richiesta per una pagina consentente PHP, il Web server invia la pagina **al motore di scripting**
 - il motore di scripting **sostituisce al codice PHP** con TAG e testo HTML e produce una pagina HTML che invia al Web server
 - il Web server invia la pagina al browser



Connessione ai DBMS

Come ci si connette ai DBMS

- si usano connettori standard o API proprietarie
- il linguaggio di scripting deve fornire le funzionalità per la connessione



PHP: un esempio

PHP e Form

```
<HTML>
<BODY>
<H3>Studenti trovati </H3>

<?php
    $nome=$_POST['nome'];
    $cognome=$_POST['cognome'];

    $Server = "segreteria.ing.unisi.it";
    $User = "pippo";
    $Passw = "pippoPass";
    $connessione = mysql_connect($Server, $User, $Passw);
    mysql_select_db("stud", $connessione);
```

lettura dei parametri

connessione al DBMS

selezione del database

PHP: un esempio II

Recupero e stampa degli studenti

esecuzione
dell'interrogazione

```
$Query = "select cognome,nome,dataNascita from studenti"
        . "where nome like %" . $nome . "% and "
        . "cognome like %" . $cognome . "%";";
```

```
$Result = mysql_query($Connessione,$query);
```

stampa
studenti

```
while ($Studente = mysql_fetch_row($Result)) {
    print "<LI>" $Studente[0] . " " . $Studente[1]. " " . $Studente[2]. "\n";
}
```

```
mysql_close($Connessione);
?>
```

```
</BODY>
</HTML>
```

chiusura della
connessione

Java Server Pages (JSP)

Cos'è

- JSP è costruito sopra alla tecnologia servlet
- una pagina JSP contiene:
 - codice HTML
 - codice Java incluso in tag specifici

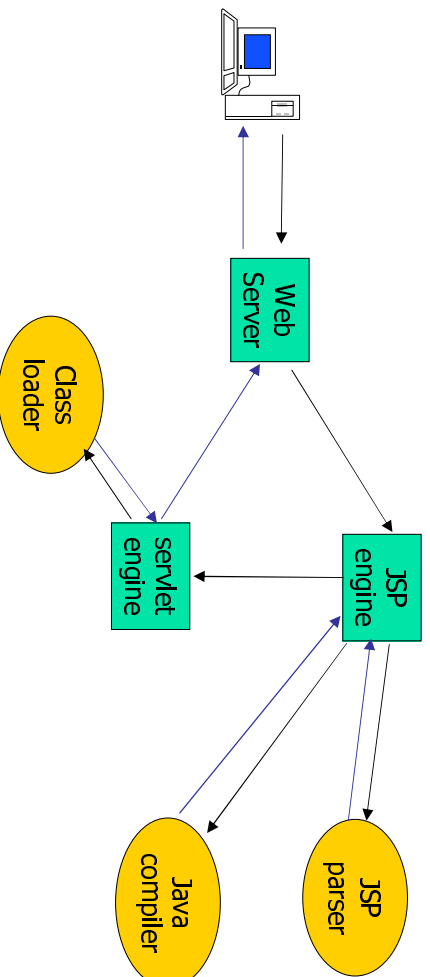
```
<% String nome = request.getParameter("nome");
String cognome = request.getParameter("cognome");
ResultSet rs = stmt.executeQuery("select cognome,nome,dataNascita"+
    "from studenti"+
    "where nome like %" + nome+ "% and " +
    "cognome like %" + cognome+"%");

while (rs.next()) { %
    String cognome = rs.getString("cognome");
    int nome = rs.getInt("nome");%>
    <LI> <%=cognome%> <%=nome%>
<%}>
```

Java Server Pages (JSP) II

Come funziona

1. il JSP engine **analizza** la pagina e crea un file sorgente Java
2. il file viene compilato in un class file che **contiene** una servlet
3. il servlet engine **carica** la **servlet** per l'esecuzione
4. la servlet viene eseguita e restituisce i risultati



CGI, Servlets e PHP

Vantaggi e svantaggi

- **Efficienza**
 - CGI è poco efficiente perché per ogni richiesta si crea un nuovo processo
 - PHP è meno efficiente delle servlets perché **completamente** interpretato, mentre le Servlets e JSP usano una codifica intermedia
- **Portabilità'**
 - CGI è uno standard, ma le applicazioni devono essere **ricomilate**
 - PHP e Servlets sono entrambe portabili, ma in PHP la connessione con DBMS può avvenire attraverso **API diverse**
- **Estensibilità'**
 - Java è un linguaggio che dispone di **numerosa API** per la connessione a DBMS (JDBC), l'interoperabilità fra processi distribuiti (RMI e CORBA), ...



CGI, Servlets e PHP

- Gestione delle sessioni
 - CGI usa i cookies per tenere traccia delle sessioni. Comunque, ad ogni richiesta occorre **riaprire la connessione** con il DBMS
 - Servlets e PHP possono mantenere aperta la connessione fra una richiesta e l'altra
- Sicurezza
 - Le servlets possono usare la **sicurezza** e la **tipizzazione esistenti in Java**
- Semplicità'
 - Il PHP e JSP forniscono un metodo molto rapido per scrivere pagine dinamiche



Application servers

Cosà sono

- sono server **posizionati fra il Web server e DBMS**
- gestiscono l'integrazione di DBMS diversi
- forniscono servizi classici come la gestione delle transazioni distribuite
- permettono la creazione e il mantenimento di oggetti
- eseguono script
- implementano tecniche di clustering e bilanciamento del carico
- possono fornire le funzionalità per la gestione di mail, accesso tramite cellulare....

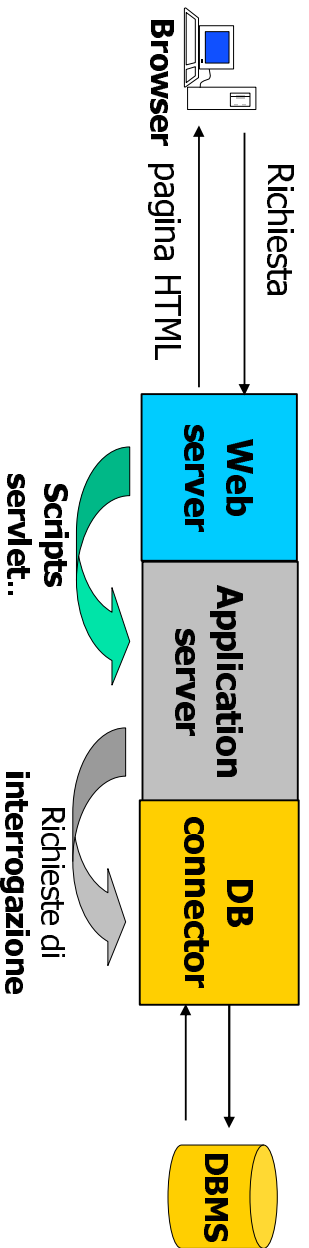
A cosa servono

- **implementano la business logic**
- facilitano l'integrazione di risorse distribuite
- vantaggiosi in grandi aziende

Application servers II

Come funzionano

- il Web server controlla il tipo di documento richiesto dal browser
 - se si tratta di una semplice pagina HTML, **provvede ad inviarla direttamente**
 - se si tratta di una pagina dinamica supportata dall'applicazione server, **invia la richiesta a quest'ultimo**
- L'applicazione server **elabora la pagina dinamica**
 - L'elaborazione può comportare l'interazione con oggetti distribuiti e DBMS



Application servers: un esempio

Oracle Application Server (IAS) contiene

- Una JVM che supporta CORBA, Enterprise Java Beans (EJB...)
- un servlet engine
- un JSP engine
- un interprete perl
- un motore per PSP (il linguaggio proprietario PL/SQL Server Pages)
- un servizio di cache
- una estensione di Apache che permette anche connessioni sicure (mod_ssl) e invia ai vari motori le pagine contenente script

Java Applet

Sono applicazioni JAVA

- compilate e memorizzate in forma di bytecode sui server
- possono essere scaricate ed **eseguite dal browser**
- si connettono ai DBMS attraverso
 - **JDBC** (Java Database Connectivity)
 - fornisce un insieme di API con cui si può connettere ad un database SQL, inviare delle interrogazione, ricevere e analizzare le risposte
 - **SQLJ**
 - una versione di **embedded SQL** specializzato per JAVA
 - **JAVA Blend**
 - crea una connessione diretta fra **oggetti JAVA** e **oggetti del database**: si opera sulle tabelle come se fossero oggetti
 - **SQLJ** e **JAVA Blend** si appoggiano su **JDBC**

Esecuzione di un'applet: un esempio

La pagina HTML che permette di caricare l'applet

```
<HTML>
<TITLE> Pagina per lanciare l'applet</TITLE>
<BODY>
Val...
<APPLET CODE="archiviostudenti.class" WIDTH=300 Height=300>
</BODY>
</HTML>
```

Il codice dell'applet

```
public class archiviostudenti extends java.applet.Applet{
    public void init() {
        myTextArea.setText('In esecuzione ...')
    }
    ...
}
```

Chiamata dal browser per segnalare il caricamento dell'applet